LogicPro
User's Guide

Watlow Anafaze
314 Westridge Drive
Watsonville, CA 95076

Phone: (831) 724-3800
Fax: (831) 724-0320

Part No. 28002-00. Revision 3.00
September 2000

Copyright © 2000
Watlow Anafaze

Information in this manual is subject to change without notice. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form without written permission from Watlow Anafaze.

Anafaze developed package includes licensed proprietary material of Intellution, Inc.
© Intellution, Inc. 1996.

Warranty

Watlow Anafaze, Incorporated warrants that the products furnished under this Agreement will be free from defects in material and workmanship for a period of three years from the date of shipment. The Customer shall provide notice of any defect to Watlow Anafaze, Incorporated within one week after the Customer's discovery of such defect. The sole obligation and liability of Watlow Anafaze, Incorporated under this warranty shall be to repair or replace, at its option and without cost to the Customer, the defective product or part.

Upon request by Watlow Anafaze, Incorporated, the product or part claimed to be defective shall immediately be returned at the Customer's expense to Watlow Anafaze, Incorporated. Replaced or repaired products or parts will be shipped to the Customer at the expense of Watlow Anafaze, Incorporated.

There shall be no warranty or liability for any products or parts that have been subject to misuse, accident, negligence, failure of electric power or modification by the Customer without the written approval of Watlow Anafaze, Incorporated. Final determination of warranty eligibility shall be made by Watlow Anafaze, Incorporated. If a warranty claim is considered invalid for any reason, the Customer will be charged for services performed and expenses incurred by Watlow Anafaze, Incorporated in handling and shipping the returned unit.

If replacement parts are supplied or repairs made during the original warranty period, the warranty period for the replacement or repaired part shall terminate with the termination of the warranty period of the original product or part.

The foregoing warranty constitutes the sole liability of Watlow Anafaze, Incorporated and the Customer's sole remedy with respect to the products. It is in lieu of all other warranties, liabilities, and remedies. Except as thus provided, Watlow Anafaze, Inc. disclaims all warranties, express or implied, including any warranty of merchantability or fitness for a particular purpose.

Please Note: External safety devices must be used with this equipment.

Table of Contents

List of Figures. XI

List of Tables XV

1 Introduction 1

This Manual 1
Where Do I Look? 1
Conventions Used 3
Safety symbols 4

2 LogicPro Workplace 5

Menu Bar 6
File Menu 7
Edit Menu 8
View Menu 11
Tools Menu 11
UDFB (User Defined Function Block) Menu 12
Execution Menu 13
Simulation Menu 13
Options Menu 14
Windows Menu 16
Help Menu 17
Status Bar 18
Toolbars 18
Standard Toolbar 18
Language Toolbars 20
Using Standard Windows Features 22
SFC Actions 22
Ladder Rungs 22
Standard Text 23

3 Projects 25

Creating a Project 26
Opening a Project 27
Copying Projects 28
Backing Up a Project 29
Closing a Project 31
Deleting a Project 31

4 Resources 33

Creating a Resource 33
Editing a Resource 36
Copying Resources 36
Resource Backup 37
Deleting a Resource 38
Opening a Resource 39

5 Programs 41

Creating a New Program 41
Opening a Program 42
Saving a Program 43
Closing a Program 43
Copying Programs 44
Saving a Program with a New Name (Save As) 45
Backing Up a Program 45
Deleting a Program 47

6 Variables 49

Naming Conventions 49
Retentive Variables 50
System Variables 50
Project Variables 51
Adding a Project Variable 52
Editing a Project Variable 53
Deleting a Project Variable 54
Cloning a Project Variable 55
Resource Variables 56
Adding a Resource/Program Variable 56
Editing a Resource/Program Variable 59
Deleting Resource/Program Variables 60
Cloning a Resource or Program Variable 61
Program Variables 61
Finding and Replacing a Variable In a Program 62
Using the Find Command 62
Using the Replace Command 64
Import/Export 65
Importing an ASCII file 65
Verifying the Import Operation 67
Creating or Editing an ASCII file 67
Exporting an ASCII file 70
Verifying the Export Operation 70

7 Sequential Function Charts 71

About SFC 71
Using SFCs in LogicPro 71
SFC Toolbar 72
Selector 73
Origin Step 73
Step 73
Simultaneous Transition 74
Loop Back 77
Action 78
Comments 79
Cross Reference 79

8 Ladder Diagrams 81

Ladder Toolbar 82
Selector 83
Normally Open Contact 83
Normally Closed Contact 84
Off To On Transitional Contact 84
On To Off Transitional Contact 85
Normal Coil 85
Latched Coil 85
Unlatched Coil 86
Function Block 86
Comments 86
Cross Reference 87
Viewing the Grid 87
Creating and Editing Ladder Diagrams 88
Inserting Rungs 88
Inserting Elements 88
Inserting Function Blocks 89
Connecting Function Blocks to Other Ladder Elements 90
Inserting Parallel (Or) Branches 93
Inserting Multiple Output Coil branches 96
Inserting Elements From the Left Power Rail 97
Cutting Rungs 98
Copying Rungs 99
Pasting Rungs 99
Deleting Elements 100
Moving Elements 100
Assigning Labels to Rungs 100
Assigning Variable Names 101
Naming and Assigning Variables to Function Blocks 102
Adding Comments to Ladder Diagrams 104
Hiding Rung Comments 104

9 Function Block Diagrams 105

FBD Toolbar 105
Selector 106
Straight Connections 106
Inverted Connections 106
Variables 107
Function Blocks 107
Cross Reference 107
Creating an FBD Program 107
Inserting Function Blocks 108
Inserting Variable Elements 109
Connecting Inputs and Outputs 109
Assigning Variables to Variable Elements 110
Assigning Names and Execution Orders to Function Blocks 111
Cutting, Copying, Pasting, and Deleting 112

10 Function Blocks 115

Standard Function Blocks 115
Timing and Counting Elements 115
Counter Up (CTU) 116
Counter Down (CTD) 117
Counter Up Down (CTUD) 119
Timer Pulse (TP) 121
Timer On-Delay (TON) 123
Timer Off-Delay (TOF) 124
Comparison 126
Greater Than (GT) 126
Less Than (LT) 127
Equal To (EQ) 128
Arithmetic 130
Add (ADD) 130
Subtract (SUB) 131
Multiply (MULT) 132
Divide (DIV) 133
Data Manipulation 135
First In - First Out (FIFO) 135
Last In - First Out (LIFO) 137
Bit Shift (SL) 140
Scan Time Counter (SCAN) 142
Move (MOVE) 143
Scale (SCL) 144
Real Number Function Blocks 145
Real Greater Than (RGT) 145
Real Less Than (RLT) 146
Real Equal To (REQ) 147
Real Add (RADD) 148
Real Subtract (RSUB) 149
Real Multiply (RMUL) 151
Real Divide (RDIV) 152
Real Move (RMOV) 153
Integer To Real (ITOR) 154

Real to Integer (RTOI) 155
Square Root (SRT) 155
Logical 157
Logical And (AND) 157
Logical Or (OR) 157
Vendor Provided Function Blocks 158
Comparison (CMP) 158
Calculation (CALC) 159
Jump (JMP) 161
InterLock (IL) 162
InterLock-Clear (ILC) 162
Rotation (RROT/LROT) 163
Shift (RSFT/LSFT) 164
Negation (NEG) 1 165

11 User Defined Function Blocks. 167
Overview 167
Getting Started 167
Defining a New or Editing an Existing UDFB 168
In's, Out's and Internals 169
UDFB Files 172
Compiling and Building a Library 174
UDFB in Simulation 175
Working with UDFB Files 177
Editing UDFB Files 177
Opening UDFB Files 177
Closing UDFB Files 177
Saving UDFB Files 178
Deleting UDFBs 178
UDFB Tutorial 179

12 Input/Output Drivers. 185
Variable Types 186
Input/Output 187
IO Size 187
PPC-2000 IO Drivers 188
IO Driver Choice 188
IO Physical Addresses 188
Using the IO Drivers 192
CPC400 IO Drivers 208
IO Driver Choice 208
IO Physical Addresses 208
Using the IO Drivers 210

13 Compiler Setup 221
Setting up the Compiler 221

14 Simulation 225

Building a Program for Simulation 226
Using the Simulation Tools 227
Simulation Setup 228
Timer Ticks 229
Increment Simulation Timer Ticks 230
Scan Simulation Logic 230
Number of Scan 230
Resetting the Simulation 231
Change Simulation Variables 231
Change Simulation Constants 232
Viewing Individual Steps and Actions within a Simulation 233
Closing the Simulation 233

15 Downloading and Monitoring 235

Executable 235
Downloading to a controller through an RS-232 Port 235
Monitoring a Program 237
Monitoring Sequential Function Charts 239
Monitoring Ladder Diagrams 239
Monitoring Function Block Diagrams 240
Forcing I/O 240
Variable Watch 241
Configuring Variable Watch 242
Selecting and Viewing Variables 242
Changing the Value of a Variable 243
Forcing Variables 243
Watch Grid View 244
On-Line Constant Changes 246

16 Cross-Referencing 249

Using Cross-Referencing 249
SFC On-line Cross-Reference 249
Ladder On-line Cross-Reference 250
Function Block Diagram On-line Cross-Reference 251
Cross-Reference Output 251
Output Printing 253
Report Setup 254
Variables Setup 255
Ladder Setup 255
SFC Setup 256
FBD Setup 257

17 Dynamic Data Exchange 259
How to Access a Remote Item 259
Installation 260
Starting the LogicPro DDE Application 260
Configuration 260
Creating the Application FPR File 261
Topics 262
Creating A New Topic 262
Open a Topic 263
Close a Topic 263
Modify a Topic 263
Delete a Topic 263
Item Point Naming 264
Finding The Available Item Names 264
Excel Spreadsheet DDE Sample 264
Setting Up Excel to Read 266

Glossary 269

List of Figures

1 Introduction 1

2 LogicPro Workplace 5

Figure 2.1—The LogicPro Workplace 6
Figure 2.2—Two Views of the LogicPro Standard toolbar, showing various active buttons 18

3 Projects 25

Figure 3.1—Projects Hierarchy 25
Figure 3.2—New Project Dialog Box 26
Figure 3.3—Open Project Dialog Box 27
Figure 3.4—Rebuild Project List Dialog Box 28
Figure 3.5—Copy Project Dialog Box 30
Figure 3.6—Delete Project Dialog Box 32

4 Resources 33

Figure 4.1—New Resource Dialog Box 33
Figure 4.2—PPC-2000 Attributes Dialog Box 34
Figure 4.3—Edit Resource Dialog Box 36
Figure 4.4—Copy Resource Dialog Box 38
Figure 4.5—Delete Resource Dialog Box 39
Figure 4.6—Open Resource Dialog Box 39

5 Programs 41

Figure 5.1—New Program Dialog Box 42
Figure 5.2—Open Program Dialog Box 43
Figure 5.3—Verify Dialog Box 44
Figure 5.4—Copy Program Dialog Box 46
Figure 5.5—Delete Program Dialog Box 47

6 Variables 49

Figure 6.1—Project Variables Dialog Box 51
Figure 6.2—Add Project Variable 52
Figure 6.3—Edit Project Variable Dialog Box 54
Figure 6.4—Resource Variable Dialog Box 57
Figure 6.5—Program Variable Dialog Box 57

Figure 6.6—Add Resource Variable Dialog Box 58
Figure 6.7—Program Variables Dialog Box 62
Figure 6.8—Find Variable 63

7 Sequential Function Charts 71

Figure 7.1—Simultaneous Transition. 74
Figure 7.2—Parallel and / or Single Transitions 75
Figure 7.3—Simultaneous Transition Conditions. 75

8 Ladder Diagrams 81

Figure 8.1—Element without Handles 83
Figure 8.2—Selected Element with Handles 83
Figure 8.3—Rungs with Multiple Branches 88
Figure 8.4—Function Block Dialog Box 89
Figure 8.5—Rung Before Adding a Function Block 90
Figure 8.6—Rung With a Function Block Added 90
Figure 8.7—Ladder Diagram Before Elements Are Added to Function Block 91
Figure 8.8—Ladder Diagram After Elements Are Added to Function Block 91
Figure 8.9—Inserting an Input Element on a Function Block 92
Figure 8.10—Element Inserted Between Left Power Rail and Second Input Bit 92
Figure 8.11—Connecting an Element to a Function Block's Output 93
Figure 8.12—Function Block After Connecting Output Element 93
Figure 8.13—Adding the Parallel Branch. 94
Figure 8.14—After Adding the Parallel Branch 94
Figure 8.15—Adding Another Branch 95
Figure 8.16—Branch Added 95
Figure 8.17—Creating a New Parallel Branch from the Left Power Rail 96
Figure 8.18—After Creating a New Parallel Branch from the Left Power Rail. 96
Figure 8.19—Before Creating Multiple Output Coil Branches 97
Figure 8.20—After Creating Multiple Output Coil Branches 97
Figure 8.21—Multiple Output Coil Branches
after the Top Coil is Removed 97
Figure 8.22—Adding an Extension from the Left Power Rail. 98
Figure 8.23—After Adding an Extension from the Left Power Rail 98
Figure 8.24—Variables Dialog Box 101
Figure 8.25—Assign Function Block Symbolic Input/Output Dialog Box 103
Figure 8.26—Symbolic Name on Function Block in a Ladder Diagram Program 103
Figure 8.27—Rung Comment Dialog Box 104

9 Function Block Diagrams 105

Figure 9.1—Function Block Dialog Box 108
Figure 9.2—Placed Function Blocks 109
Figure 9.3—Symbolic Name on Function Block in a FBD program 110
Figure 9.4—Assign Name to Function Block Dialog Box 111
Figure 9.5—Execution Order in FBD 112
Figure 9.6—Function Blocks in FBD with Handles 113
Figure 9.7—Pasted Function Block in FBD 113
Figure 9.8—Function Block Before Deletion, With Handles 114
Figure 9.9—Function Block After Deletion. 114

10 Function Blocks 115

Figure 10.1—Counter Up Function Block 116
Figure 10.2—Assign Function Block Symbolic Input/Output Dialog Box. 117
Figure 10.3—Counter Down Function Block 118
Figure 10.4—Counter Up Down Function Block 119
Figure 10.5—Timer Pulse Function Block 121
Figure 10.6—Relationship between the Input and Output of TP Function Block 123
Figure 10.7—Timer On-Delay Function Block 123
Figure 10.8—Relationship Between Inputs and Outputs of TON Function Block 124
Figure 10.9—Timer Off-Delay Function Block. 125
Figure 10.10—Relationship Between Inputs and Outputs of TOF Function Block . . . 126
Figure 10.11—Greater Than Function Block. 126
Figure 10.12—Less Than Function Block 128
Figure 10.13—Equal To Function Block 129
Figure 10.14—ADD Function Block 130
Figure 10.15—Subtract Function Block. 131
Figure 10.16—Multiply Function Block 132
Figure 10.17—Divide Function Block 134
Figure 10.18—First In - First Out Function Block 135
Figure 10.19—Last In - First Out Function Block 138
Figure 10.20—Bit Shift Function Block 140
Figure 10.21—Example of How Bits Shift Through the SL Function Block. 141
Figure 10.22—Scan Time Counter Function Block 142
Figure 10.23— Move Function Block 143
Figure 10.24— Scale Function Block 144
Figure 10.25—ROUT as a function of INP 145
Figure 10.26— Real Greater Than Function Block 146
Figure 10.27— Real Less Than Function Block 147
Figure 10.28— Real Equal To Function Block 148
Figure 10.29— Real Add Function Block 149
Figure 10.30— Real Subtract Function Block 150
Figure 10.31— Real Multiply Function Block. 151
Figure 10.32— Real Divide Function Block. 152
Figure 10.33— Real Move Function Block 153
Figure 10.34— Integer to Real Function Block 154
Figure 10.35— Real to Integer Function Block 155
Figure 10.36— Square Root Function Block. 156
Figure 10.37— The Logical AND Function Block 157
Figure 10.38— The Logical OR Function Block 157
Figure 10.39—Vendor Provided Function Block diagram 158
Figure 10.40— The Comparison Function Block. 159
Figure 10.41— The Calculation Function Block 160
Figure 10.42— The Jump Function Block 161
Figure 10.43— The InterLock Function Block 162
Figure 10.44— The InterLock Clear Function Block 162
Figure 10.45— The Right and Left Rotation Function Blocks 163
Figure 10.46— The Right and Left Shift Function Blocks 164
Figure 10.47— The Negation Function Block 165

11 User Defined Function Blocks. 167

Figure 11.1—New UDFB Dialog Box 169
Figure 11.2—Add Symbol Definition Dialog Box. 171

Figure 11.3—Edit Symbol Definition Dialog Box 171
Figure 11.4—Build UDFB Library for Execution 174
Figure 11.5—Build UDFB Library for Simulation Dialog Box 176
Figure 11.6—Open User Defined Function Block Dialog Box 177
Figure 11.7—Delete User Defined Function Block Dialog Box 178
Figure 11.8—ADD1 Function Block 180

12 Input/Output Drivers 185

13 Compiler Setup 221
Figure 13.1—Simulation Compiler 221
Figure 13.2—Compiler Paths 222
Figure 13.3—Select Borland Compiler 222

14 Simulation 225
Figure 14.1—Compiler Dialog Box 227
Figure 14.2—Borland C++ Compiler DOS window 227
Figure 14.3—Simulation Setup Dialog Box 229
Figure 14.4—Change Variables Dialog Box 231
Figure 14.5—On-Line Function Block Constant Change 232

15 Downloading and Monitoring 235
Figure 15.1—Downloading 237
Figure 15.2—LogicPro Monitoring Window 238
Figure 15.3—Active Step in an SFC 239
Figure 15.4—Monitoring Ladder Diagrams 240
Figure 15.5—Force - Clear IO Dialog Box 241
Figure 15.6—The Watch Grid in the Variable Watch Window 245
Figure 15.7—On-Line Function Block Variables Change Dialog Box 247

16 Cross-Referencing 249
Figure 16.1—The Cross-Reference Pointer 250
Figure 16.2—Print Dialog Box 253
Figure 16.3—Report Setup Dialog Box 254
Figure 16.4—Variable Setup Dialog Box 255
Figure 16.5—Ladder Setup Dialog Box 256
Figure 16.6—SFC Setup Dialog Box 256
Figure 16.7—FBD Setup Dialog Box 257

17 Dynamic Data Exchange 259
Figure 17.1—Example 1 265
Figure 17.2—Spreadsheet Example 266

List of Tables

1 Introduction 1

Table 1.1—Chapter Descriptions 2
Table 1.2—Conventions Used. 4

2 LogicPro Workplace 5

Table 2.1—The File Menu 7
Table 2.2—The Edit Menu. 9
Table 2.3—The View Menu 11
Table 2.4—The Tools Menu 12
Table 2.5—The UDFB Menu 13
Table 2.6—The Execution Menu 13
Table 2.7—The Simulation Menu 14
Table 2.8—The Options Menu 14
Table 2.9—The Windows Menu 16
Table 2.10—Levels of Help 17
Table 2.11—Help Menu Options 17
Table 2.12—Toolbar Definitions 18
Table 2.13—The SFC Toolbar 20
Table 2.14—The Ladder Toolbar 21
Table 2.15—The FBD Toolbar 21

3 Projects 25

4 Resources 33

5 Programs 41

6 Variables 49

Table 6.1—File Format Specification of Import/Export 68
Table 6.2—Selected Explanations of the Example Import/Export File . . . 69

7 Sequential Function Charts 71

Table 7.1—The SFC Toolbar 72
Table 7.2—Transition Condition Expressions 76

8 Ladder Diagrams 81
Table 8.1—Ladder Diagrams Toolbar 82

9 Function Block Diagrams 105
Table 9.1—FBD Toolbar Functions 106

10 Function Blocks 115
Table 10.1—Symbol Operation 160

11 User Defined Function Blocks 167

12 Input/Output Drivers 185
Table 12.1—Variable Types 186
Table 12.2—Input and Output 187
Table 12.3— IO Sizes Used with IO Drivers 187
Table 12.4—Data Accessed by the IO Drivers 188
Table 12.5—Categories of IO Drivers 189
Table 12.6—Address Syntax for Module Specific Drivers 189
Table 12.7—I/O Type Digit in IO Physical Address 190
Table 12.8—Addresses Used with Parameter Specific IO Drives 190
Table 12.9—Address Syntax for Database Driver 191
Table 12.10—Data Accessed by the IO Drivers 208
Table 12.11—Categories of IO Drivers 208
Table 12.12—Addresses Used for Hardware Specific Drivers 209
Table 12.13—Addresses Used with Parameter Specific IO Drivers 209
Table 12.14—Address Syntax for Database Driver 210

13 Compiler Setup 221

14 Simulation 225
Table 14.1—Simulation Tools 228

15 Downloading and Monitoring 235
Table 15.1—The Columns in the Watch Grid 245

16 Cross-Referencing 249

17 Dynamic Data Exchange 259

1

Introduction

Welcome to *LogicPro*, a Microsoft Windows based graphical programming environment that allows you to create and debug programs, and manage real-time distributed or embedded control systems. *LogicPro* allows you to graphically write a “C” program without knowing the language.

LogicPro use the familiar conventions of typical graphical programming languages like *Grafcet, Ladder Logic,* etc. These languages have evolved into proven, effective solutions for automation and control projects.

This Manual

If it relates to *LogicPro*, you should be able to find it between these covers with few exceptions.

Where Do I Look?

Take some time familiarizing yourself with the layout and design of this manual. Refer to *Table 1.1 on page 2* for a complete listing of the section numbers and titles along with a general overview of what you can expect to find in each chapter.

Table 1.1 Chapter Descriptions

Chapter	Title	Description
1	Introduction	Introduction and general information.
2	The *LogicPro* Workplace	The *LogicPro* Workplace and screen layout.
3	Projects	Creating, using, and changing *LogicPro* Projects.
4	Resources	Creating, using, and changing *LogicPro* Resources.
5	Programs	Creating, using, and changing *LogicPro* Programs within a resource.
6	Variables	Creating, using, and changing *LogicPro* Variables within projects, resources, and programs.
7	Sequential Function Charts	Explanation of the tools available for programming in the *SFC* (Sequential Function Chart) language in *LogicPro*.
8	Ladder Diagrams	Explanation of the tools available for programming in the Ladder Diagram (LD) language in *LogicPro*.
9	Function Block Diagrams	Explanation of the tools available for programming in the *FBD* (Function Block Diagram) language in *LogicPro*.
10	Function Blocks	Information on Standard IEC 1131-3 and Vendor provided function blocks: specific information on many of the most common function blocks, how to use function blocks in your programs, and where to look for more information.

Chapter	Title	Description
11	User Defined Function Blocks	Information about creating and using User Defined Function Blocks.
12	I/O Drivers	Information on input/output drivers, examples, and where to look for more information.
13	Compilers	Information about using the Borland C++ compiler.
14	Simulation	Information on the simulator function of *LogicPro*, specifically, how to use it.
15	Downloading and Monitoring	How to download a program to your hardware and monitor it once it's been downloaded.
16	Cross-Referencing	How to use the Cross-Referencing functions of *LogicPro*.
17	Dynamic Data Exchange	Using DDE for accessing and editing *LogicPro* data.
	Glossary	Definitions of the terms, acronyms, and expressions used in this manual.
	Index	Where to find the information you're looking for.

Conventions Used

To help improve the readability of this manual we use different fonts and styles to identify specific actions you will initiate in the program.

Table 1.2 on page 4 lists the conventions used as well as a brief description of each.

Table 1.2 Conventions Used

Convention	Explanation
Italics	Indicates references to *Chapters, sections, Figures* and *Tables*.
Capitalization	Indicates the proper name of an element of *LogicPro*.
Bold Sans Serif	Indicates *LogicPro* menu headings or options.
[Bold San Serif in Square Brackets]	Indicates the name of a button or tool on the screen.
<Bold serif text in Angular Brackets>	Indicates a specific key or combination of keys on your keyboard.
ALL CAPS, SANS SERIF, MONOSPACE	Indicates text as it appears in function blocks in the program editor.
Unbolded, serif, monospace	Indicates a filename or code as it appears in the User-Defined Function Block programming windows, or text as it appears in ASCII files.
Unbolded Sans Serif	Sample cross-reference print out.

Safety symbols

These symbols are used throughout this manual:

WARNING! ***Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury.***

CAUTION! ***Indicates a potentially hazardous situation which, if not avoided, could result in minor or moderate injury or property damage.***

NOTE! ***Indicates pertinent information or an item that may be useful to document or label for later reference.***

2

LogicPro Workplace

LogicPro uses the Microsoft Windows' Graphical User Interface (GUI) as its operating environment. By using standard Windows, the *LogicPro* environment allows you to easily accomplish development tasks in an environment that you are already familiar and comfortable with.

To successfully use *LogicPro*, you should know how to use Windows. The following sections provide a description of *LogicPro's* screen elements, as well as a reference to some of the more common Windows features you will use. For additional information, please refer to your Microsoft Windows documentation.

The Workplace is made up of several different Screen Elements. *Figure 2.1 on page 6* illustrates the *LogicPro* workspace and identifies the following elements:

- Menu Bar
- Status Bar
- Standard Toolbar
- Language Toolbars

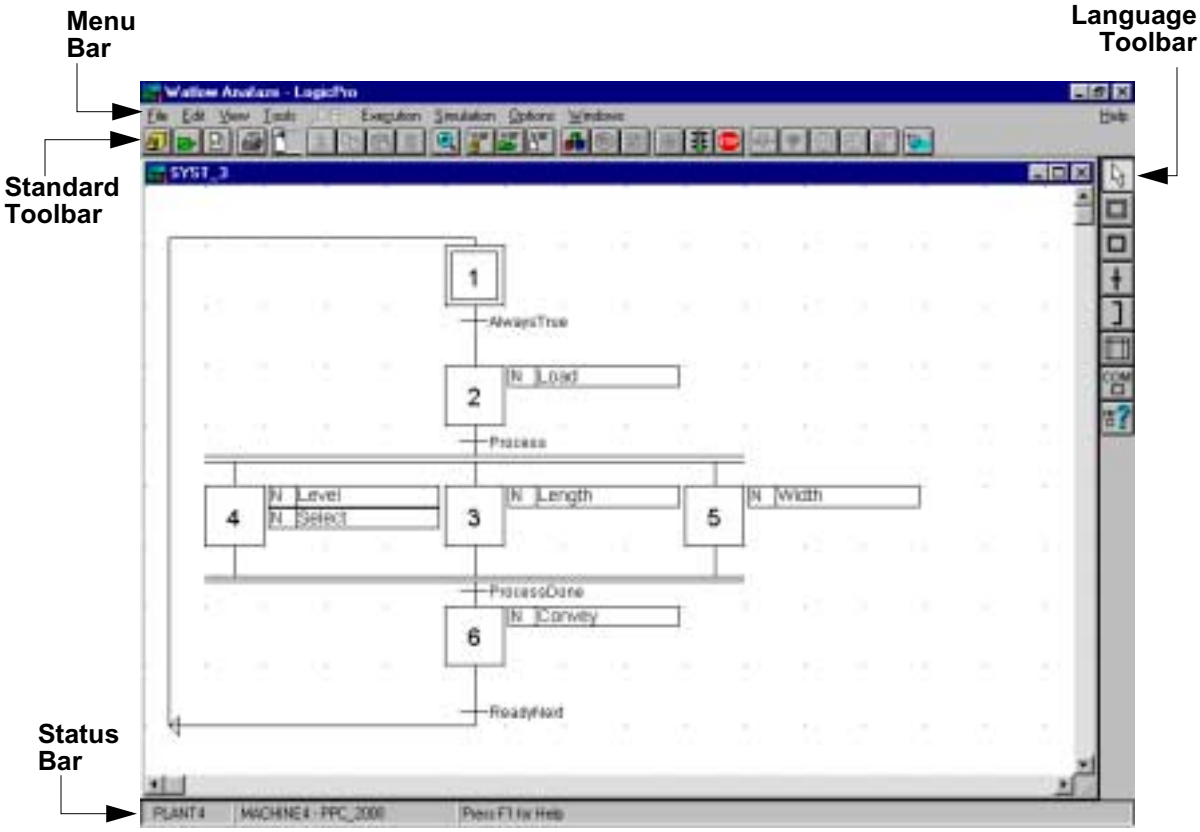


Figure 2.1 The LogicPro Workplace

Menu Bar

There are two ways to choose an item from a menu:

- With the mouse, click a menu then click the item.
  – or –
- Press the **<ALT>** key, press the underlined letter in the menu's name, and press the underlined letter in the item's name.

There are ten (10) menus available:

- **File**
- **Edit**
- **View**
- **Tools**
- **UDFB**
- **Execution**
- **Simulation**
- **Options**

- **Windows**
- **Help**

LogicPro enables the appropriate menus. Each of these menus allows control of a specific part of *LogicPro*.

File Menu

This menu provides easy access to most tools you will need for creating, opening and saving files and printing reports. Additionally, tools for Importing and Exporting variables are available. See *Table 2.1* for a detailed listing of menu items.

Table 2.1 The File Menu

Menu Option	Description
New	Creates a new file type from the menu: Project… Resource… Program… User Defined Function Block…
Open	Opens a file type from the menu: Project… Resource… Program… User Defined Function Block…
Close	Closes the chosen type of item: Project… Resource… Program… User Defined Function Block… Simulation Monitoring
Delete	Deletes the chosen type of item: Project… Resource… Program… User Defined Function Block…
Save	Saves the program in the active (top) window.
Save As…	Saves the information in the active window under a different file name.

Menu Option	Description
Backup	Creates a backup copy of the chosen type of item: Project… Resource… Program…
Import Variable	Reads data from an ASCII, comma-delimited file. Allows you to create and edit resource and program variables externally to *LogicPro*.
Export Variable	Writes data to an ASCII, comma-delimited file. Allows you to edit resource and program variables external to *LogicPro*.
Report Setup…	Allows you to customize your documentation by specifying what information appears in the printout.
Print Setup…	Customize user defined print options: Which printer to print on. Size and orientation of the paper, and how to print (single-sided or double-sided if your printer can print double-sided). Printer resolution, intensity, and graphics mode. The kinds of fonts to use while printing. Things like print quality and memory tracking for the printer.
Print…	Prints selected portion of an open project or program.
Exit	Closes all open projects and shuts down *LogicPro*.

Edit Menu

This menu provides useful tools for managing changes for the data in the active window. See *Table 2.2 on page 9* for a detailed listing of menu items.

Table 2.2 The Edit Menu

Menu Option	Description
Undo	**Note:** *This option is only available while editing UDFB C/C++ source (*.c) and header (*.h) files.* Also, note that the **Undo** buffer is only one level deep: it only contains the last keystroke. Selecting **Undo** twice will perform a 'double-undo', or redo. Restores what you are working on to the way it was before your very last action. For example if you overwrite a line of code, this command will remove your edits and restore the line as it was before you began.
Cut **<Ctrl X>**	Removes the selected objects and places them in a temporary file. This material remains in the temporary file until overwritten by subsequent **Cut** or **Copy** actions, or until you close *LogicPro*. The **Cut** feature works the same way when working with UDFBs, except that the data is stored on the Windows Clipboard, not the LogicPro temporary file.
Copy **<Ctrl C>**	Creates a duplicate of the selected object without removing it (see note below), and places it in a temporary file. This material remains in the temporary file until overwritten by subsequent **Cut** or **Copy** actions, or until you close *LogicPro.* The **Copy** feature works the same way when working with UDFBs, except that the data is stored on the Windows Clipboard, not the *LogicPro* temporary file.
Paste **<Ctrl V>**	Places the content of the temporary file into the active window at the insertion point. If you are pasting in a UBFB, it is the content of the clipboard that is placed in the active window. Pasting does not clear the temporary file. The last item placed into it stays there until subsequent **Copy** or **Cut** actions, or until you close *LogicPro.*
Find…	Searches the active window for a specific variable.
Replace…	Searches the active window for every instance of a specific value and replaces it with a new, user-defined value.

Menu Option	Description
Clear Del	Deletes the selected area from the active window. This operation cannot be undone.
Variables	Opens a dialog box and allows you to manage the different kinds of variables. Select the variable type from the following list: **Project…** **Resource…** **Program…**
Attributes	Changes the attributes of an object within an existing project. Modifies the different configurations dependent on your selection from the following submenu: **Resource…** **Driver…** **Program…** **Action…** **User Defined Function Block…**

NOTE!

Changes to variables within LogicPro (such as* Add, Edit, *or* Delete*) take place immediately and do not depend on a Program save.

Cut, Copy *and* Paste *functions are available primarily with text and cannot be used with most graphical program objects. To test if an object can be cut, copied or pasted, select it then select the Edit menu item to see if the* Cut *and* Copy *selections are available.*

You can only paste cut or copied elements or text into a window of the same type. For example, if you copy a rung from a ladder diagram you can only paste it into a ladder diagram. LogicPro disables the paste function if you are in a different language type window.

View Menu

This menu allows you to customize the appearance of your workspace. See *Table 2.3* for a detailed listing of each menu item. An enabled selection is easily identified by a checkmark to the left of a menu item.

Table 2.3 The View Menu

Menu Option	Description
Zoom…	Changes the magnification for the active window from 10% - 200%. This option allows you to view as little or as much of the active window as you desire. **Zoom** settings are only kept while the window is open. The next time you open the window the zoom setting reverts to the default 100% setting.
Repaint	Redraws the active window and displays the latest changes recorded in the system.
Grid	Superimposes a grid pattern over the workspace as an aid to placing items.
Comments	Allows you to toggle ladder rung comments on and off in the active window.
Monitor Program	Allows you to see the real-time status of the downloaded PLC program.
Monitor On/Off	Allows you to selectively suspend real-time monitoring messages to a PLC program on a window by window basis.
Configuration	Displays a partial IEC 1131-3 configuration descriptor.

Tools Menu

This menu contains the tools associated with your choice of programming language. Choose your programming language in the first menu item. The other available options are driven by this choice. See *Table 2.4 on page 12* for a detailed listing of the languages and their options.

Table 2.4 The Tools Menu

Menu Option	Description
Language	Select the programming language for the active window from the submenu: **SFC - Grafcet** Sequential Function Chart language. This is the *LogicPro* default language for any program window. You may change the defaults using the Options menu. **LD - Ladder** This option selects the Ladder Diagram language for the active window. **FBD -** This option selects the Function Block Diagram language for the active window.
Tools	The selected language determines the options for the rest of this menu. The options displayed correspond directly with the tools available from the different Language Toolbars. For more information, please refer to the appropriate section concerning the language you wish information on. **NOTE:** *Once you select a language and start to build a program, LogicPro prevents you from changing the language as long as any element exists in the window.*

UDFB (User Defined Function Block) Menu

This menu gives you control over the various tools available when you're creating or editing a UDFB. There are compile and build options available for both execution and simulation purposes because each creates its own variant of the UDFB and saves it in the corresponding library. See *Table 2.5 on page 13* for a detailed list and explanation of the items on the UDFB menu.

Table 2.5 The UDFB Menu

Menu Option	Description
Compile for Execution	Compiles and error checks a UDFB in the active window for downloading, using the compiler selected for the particular resource.
Build Execution Lib…	Performs the same function as Compile for Execution (above), but also adds the `.OBJ` file to the UDFB library for this resource type. You must add a UDFB to the library before you can use it in a program.
Compile for Simulation	Compiles and error checks the UDFB in the active window for simulation using the Borland® C++ compiler.
Build Simulation Lib…	Performs the same function as Compile for Simulation (above), but also adds the `.OBJ` file to the UDFB library for this resource type. You must add a UDFB to the library before you can use it in a program.

Execution Menu

This menu contains many useful tools that you will need to compile, link, download, stop, and start a project. See *Table 2.6* for a detailed list and explanation of the items on the Execution menu.

Table 2.6 The Execution Menu

Menu Option	Description
Build All and Run	Compiles, downloads and starts the program.
Download and Run	Downloads and starts a previously compiled program.
Start Executable	Starts a previously downloaded program.
Terminate	Stops the currently running program.

Simulation Menu

This menu gives you control over all aspects of running a simulation. Refer to *Table 2.7 on page 14*.

Table 2.7 The Simulation Menu

Menu Option	Description
Build All and Run	Compiles and links a program for simulation and runs the simulation.
Simulation Setup…	Opens the Simulation Setup dialog box, allowing you to set the number of milliseconds that elapse per scan, as well as establishing the number of logic scans per simulation scan.
Simulation Reset	Resets all variables and the program state back to their original values.
Scan	Executes a single scan of the simulation based on the simulation setup criteria. This feature is only available if you are not in the continuous scan mode.
Increment Timer	Advances the timer by the number of "ticks" defined in the **Simulation Setup** dialog box.
Change Variable…	Opens the **Change Variables** dialog box, allowing you to change the value of any variable in the program for simulation purposes.

Options Menu

This menu allows you to customize the *LogicPro* development environment. To activate an option, select the menu item. A check mark appearing to the left of an item indicates the option is active. If there is no check mark, the option is inactive. See *Table 2.8*.

Table 2.8 The Options Menu

Menu Option	Description
Resource Compiler Selection	Allows you to establish a path to the directory containing the compiler, linker, includes, etc.

Menu Option	Description
Program Window	Customize the application workspace using the following submenu: **Display Grid by Default** - Toggles the default setting for display grid between either on or off. The grid can also be turned on or off independently on a window-by-window basis on the View menu. **Display Comments by Default** - Toggles the default setting for display comments between either on or off. Comments can also be turned on or off independently on a window-by-window basis on the View menu. **Default Language** - Selects the default language for new program windows. The options are: **SFC (default language)** **Ladder** **FBD**
Variable	Customize how variables are handled. **Confirm Deletion** - If selected, *LogicPro* prompts you to confirm your intention to make a deletion. **Confirm Edit Change** - If selected, *LogicPro* prompts you to confirm your intention to save a change. **Auto SFC Operator** - If selected, the operator dialog box opens automatically if you attempt to build a transition expression.
AutoSave	Allows you to toggle the **AutoSave** feature (on and off) and set the save interval (from 5 to 120 minutes).
FastLink Config	Establishes a connection between FastLink, the *LogicPro* data API and a single *LogicPro* project. This allows you to browse for the `.FLK` file-path establishing **Access Time, Poll Rate** and **Message Time Out**.
Fording Options	When monitoring a program, items on this menu allow you to enable, disable, or clear fording of variables. See Forcing I/O on page 240.

Windows Menu

This menu offers additional tools for customizing your *LogicPro* workspace. Specifically, it provides a list of open windows and allows you to change the dimensions and arrangement of those windows. Refer to *Table 2.9*.

Table 2.9 The Windows Menu

Menu Option	Description
Cascade	Arranges all of the open windows in an overlapping fashion so that the title bars of each are visible, with the currently active window on top.
Tile	Arranges all of the open windows next to each other. **NOTE:** *Tile displays all of the open windows equally and side-by-side within the program window. The more windows you have open when you tile, the less of each you will see.*
Arrange Icons	Organizes the icons that represent the minimized windows in your program. This option arranges the icons along the lower left-hand edge of the workspace.
Close All	Closes all the windows associated with the: Program Simulation Monitoring UDFB All Windows
(Open Window)	Currently open windows, if any, are listed. To change the active window to any of the currently open windows, simply point to the desired window on the list and click. A check-mark appears to the left of the active window.

Help Menu

The **Help** menu gives you access to the online documentation system. This allows you to obtain information about software features and functions while in the middle of a development task.

There are three levels of information in the online help system, as described in *Table 2.10*.

Table 2.10 Levels of Help

Type of Help	Access
Complete online help.	Use the Help menu.
Information about the active dialog box or pull-down menu.	Click a **Help** button or press **<F1>**.
Information about the selected menu item.	Press **<F1>**.

Help Menu Commands

Table 2.11 briefly describes each **Help** menu command.

Table 2.11 Help Menu Options

Command	Description
Index…	Accesses the *LogicPro* Help Contents. Simply click on any of the underlined topics to navigate through the **Help** system. This will display either the appropriate help topic or an index of related topics.
Search for Help On…	Lets you define a keyword to search on. The **Help** system then lists the keyword's related topics and lets you select one.
How to Use Help…	Explains how to use the online help system.
About LogicPro	Displays copyright and version information along with other product information and limits (Demo, Standard, Unlimited.)

Status Bar

The Status bar displays project, resource, and resource type names, from left to right at the bottom of the screen, and is available to the developer within the development environment. The Status bar also provides information about the currently highlighted menu item.

Toolbars

There are two different types of toolbars:

- Standard toolbar
- Language toolbars

Each contains tools designed to make creating and managing *LogicPro* projects a fast, easy and intuitive process.

Standard Toolbar

The Standard toolbar appears across the top of the screen under the menu bar. It provides quick access to common operations, such as deleting an object, adjusting the screens zoom setting, saving, monitoring, compiling, starting and stopping. ToolTips display on-screen descriptions of all toolbar buttons simply by pausing the pointer on them.

Figure 2.2 Two Views of the LogicPro Standard toolbar, showing various active buttons

Table 2.12 Toolbar Definitions

Button	Description
	Open Project
	Open Resource
	Open Program

Button	Description
	Print Documentation
	Save
	Edit Cut
	Edit Copy
	Edit Paste
	Edit Delete
	Zoom Ratio
	Edit Project Variable(s)
	Edit Resource Variable(s)
	Edit Program Variable(s)
	Build All and Run Execution
	Download and Run
	Open Program for Monitoring
	Disabled Status Update
	Run Execution
	Terminate Execution
	Simulation Setup
	Simulation Reset
	Scan Simulating Logic
	Increment Simulation Timer Tick
	Change Simulation Variable(s)
	Repaint Window

Language Toolbars

The Language toolbars appear down the right side of the screen. Tools represent each of the language elements used in building a program in a particular language. This Toolbar automatically changes to reflect the current programming language. ToolTips are available on all the buttons so that you can easily identify their functions.

Each of the tools available on these toolbars is full explained in the section covering the particular language.

Toolbar Explanation	Page
Sequential Function Charts	72
Ladder	82
Function Block Diagram	106

NOTE! ***The Language toolbar is only visible when a program editor window is open and on top.***

Table 2.13 The SFC Toolbar

Tool	Description
	Selector
	Origin Step
	Step
	Simultaneous Transition (Directed Link)
	Loop Back (Connection)
	Action
	Comments
	Cross Reference

Table 2.14 The Ladder Toolbar

Tool	Description
	Selector
	Normally Open Contact
	Normally Closed Contact
	Off to On Transitional Contact (Positive)
	On to Off Transitional Contact (Negative)
	Normal Coil
	Latched Coil (Set)
	Unlatched Coil (Reset)
	Function Block
	Comments
	Cross Reference

Table 2.15 The FBD Toolbar

Tool	Description
	Selector
	Straight Connector
	Inverted Connector
	Variable
	Function Block
	Cross Reference

Using Standard Windows Features

LogicPro provides standard Windows features. If you are already familiar with Windows, the **Select, Cut, Copy** and **Paste** tasks are familiar.

You can access the **Cut, Copy** or **Paste** commands from **Edit** on the Menu Bar, or from the buttons on the Standard toolbar. You can also access these features using the short cut keys (**<Ctrl + X>, <Ctrl + C>,** and **<Ctrl + V>** respectively).

Programming Language	Items that can be Cut, Copied or Pasted
SFC	Actions
Ladder Diagram	Ladder Rungs
UDFB	Text

SFC Actions

Cut and copy SFC actions in the following manner:

1. Click the action with the Selector tool. This creates handles around the action.
2. Use the short cut keys or menu options to either **Cut** or **Copy** the action onto the clipboard.
3. **Paste** the action into other SFC diagrams using either the **Edit** menu or short cut keys.

If you paste the action on to the same level of SFC, you must change the action name. The editor will prompt you for this change when required.

Pasted items appear at the bottom of the list. If you need the pasted section to appear higher-up in your diagram you must cut everything above it and paste it below.

Ladder Rungs

Edit Ladder rungs in the following manner:

1. Click the rung number to the left of the ladder rung with the Selector tool. Multiple rungs can be simultaneously selected, or lassoed.
2. Use the short cut keys or menu options to either **Cut** or **Copy** the selection onto the clipboard.
3. **Paste** the contents of the clipboard into any ladder diagram using either the **Edit** menu or short cut keys.

Standard Text

To **Cut, Copy** and **Paste** text from the UDFB editor:

1. Select the text you wish to copy by holding down the left mouse button while dragging it across the desired text.
2. Use the short cut keys or menu options to either **Cut** or **Copy** the selection onto the clipboard.

Paste the contents of the clipboard into any part of the UDFB editor using either the **Edit** menu or short cut keys.

3

Projects

Before you begin it is important that you understand the structure and building blocks of a *LogicPro* Project. The *Project* itself resides at the highest level of this hierarchy. The next level is called the *Resource* level. A *Resource* is the controller hardware that will run a logic program. A project may consist of many resources, or as few as one. Each resource contains one or more *Programs*. At the lowest level is the *Action.* Actions reside within the programs. The following diagram illustrates these building blocks and their relationship to each other.

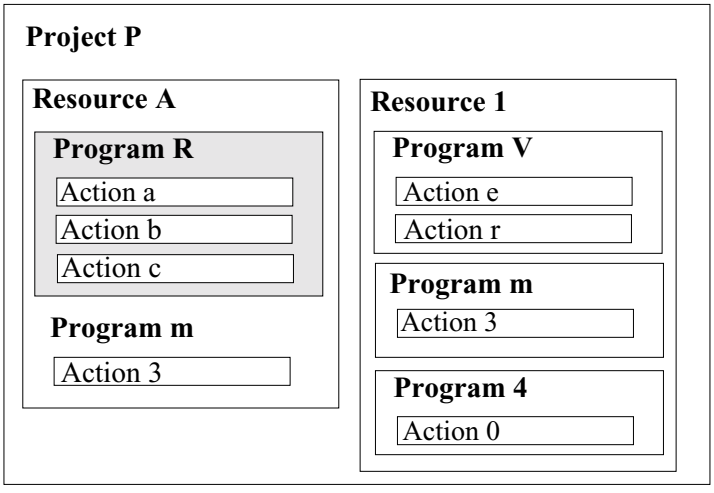


Figure 3.1 Projects Hierarchy

Each *LogicPro* project resides within a separate directory or folder. The software creates a new directory for each new project. As the developer, you can organize projects by selecting where *LogicPro* creates the directory.

The directory's name corresponds to the project's name and the contains all of the resource subdirectories. These subdirectories, in turn, contain all the resource and program files.

In *LogicPro*, the project is a receptacle for all related resources and programs. The project contains all the programmable controller elements necessary to describe and implement an automated task.

Projects can contain any number of resources and programs. This enables you to set-up Projects with numerous controllers and steps, or smaller projects with only one or two resources.

Creating a Project

Creating a Project is the first task when using *LogicPro*. You must create a Project before you can:

- Configure a controller resource.
- Design a program for a device.
- Define individual action steps.

To create a new project:

1. Select the **File** menu.
2. Select **New.**
3. Select **Project**. The **New Project** dialog box appears, as shown in *Figure 3.2*.

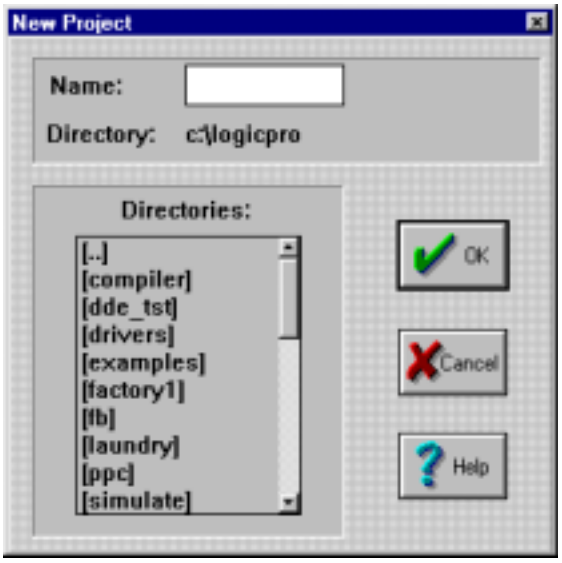


Figure 3.2 New Project Dialog Box

4. Select the directory in which you want to store your new project.
5. Enter the name for the **New Project**. The Project name must conform to standard DOS naming conventions and have no more than eight characters.
6. Click **[OK]**.

LogicPro creates a new directory on your drive. This directory contains all the files related to this Project. *LogicPro* automatically prompts you to create a New Resource. *See "Editing a Resource" on page 36* for how to create a New Resource. You cannot use resources that have been created under other Projects with your New Project.

Opening a Project

To open an existing Project:

1. Select the **File** menu.
2. Select **Open**.
3. Select **Project**. The **Open Project** dialog box appears, as shown in *Figure 3.3*.

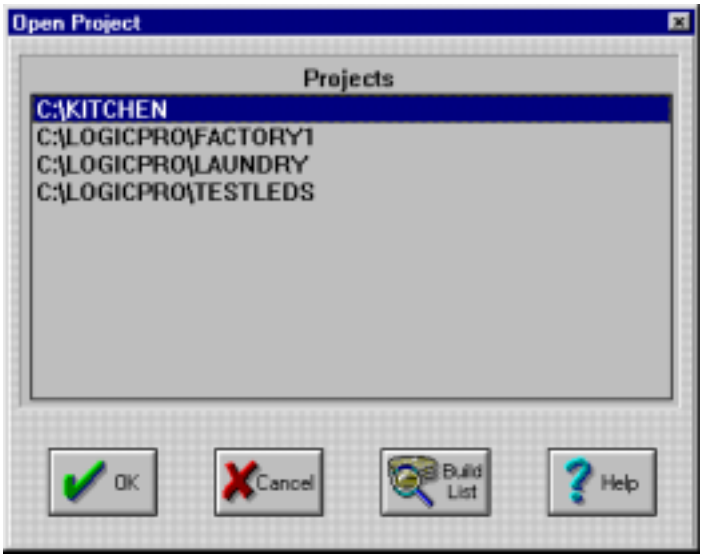


Figure 3.3 Open Project Dialog Box

4. Click on a Project in the box.
5. Click **[OK]**. *LogicPro* automatically prompts you to open a Resource and then a Program if any exist or to create them. See *Chapter 5, Programs*.
6. If you cannot find the Project you are looking for,
   a) Select **Build List** from the dialog box. The **Rebuild Project List** dialog box appears, as shown in *Figure 3.4*.
   b) Enter a drive in the **Start Drive** and **End Drive** fields. *LogicPro* searches the drive(s) entered and produces a list of all available projects.

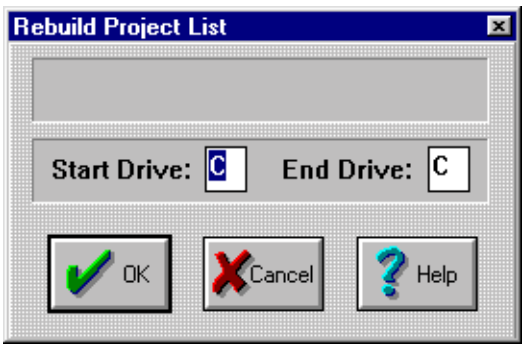


Figure 3.4 Rebuild Project List Dialog Box

Copying Projects

Projects, Resources and Programs are stored in files and directories that can be copied to a floppy or another drive location. However, the structure between these must be maintained for *LogicPro* to read files correctly. Each Project has its own directory with the same name as the project. Each Resource also has its own directory with the same name as the Resource; it is located in the associated Project directory. Programs are composed of several files which are located in the Resource directory.

To copy a Project:

1. If a Project is being used by *LogicPro*, close it.
   a) Select **File**.
   b) Select **Close**.
   c) Select **Project**.
2. Copy the Project directory using any file utility such as Windows Explorer. Copy the entire directory including all files and subdirectories. Do not rename any directory or file name; *LogicPro* will not recognize a Project in a folder if the name has been changed.
3. Open the **Project**.
   a) Select **File**.
   b) Select **Open**.
   c) Select **Project**. The **Open Project** dialog box appears, as shown in *Figure 3.3 on page 27*.
   d) The Project will not be listed. Select **Build List** from the dialog box. The **Rebuild Project List** dialog box appears, as shown in *Figure 3.4 on page 28*.
   e) Enter a drive in the **Start Drive** and **End Drive** fields. *LogicPro* searches the drive(s) entered and produces a list of all available Projects.
   f) Click on the copied Project.
   g) Click **[OK]**.

Backing Up a Project

You can back up or copy a project to any location that is accessible to the computer. Backing up is the equivalent to **Save As** for projects. To backup a project:

NOTE! ***A project must be closed before you can back it up!***

1. Select the **File** menu.
2. Select **Backup**.
3. Select **Project**. The **Copy Project** dialog box appears, as shown in *Figure 3.5*.
4. Select the Project to be backed up from the list of **Existing Project(s)**. Enter the new name for the backup copy of the project.
5. Select the drive and directory in which *LogicPro* should place the copy.

NOTE! ***If you are backing up the project to protect it from mechanical failure, then it should be stored on a different disk drive or storage media.***

6. Click **[OK]**.

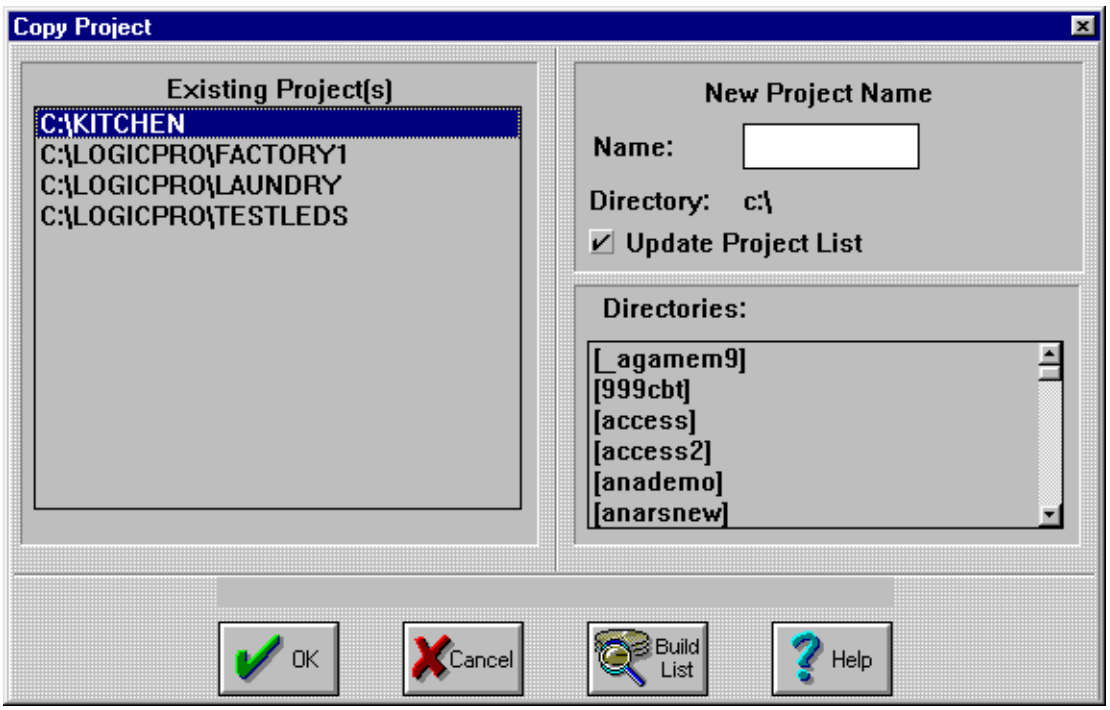


Figure 3.5 Copy Project Dialog Box

Closing a Project

To close a Project:

1. Select the **File** menu.
2. Select **Close**.
3. Select **Project**.

If any of the Programs within the Project you are closing have changed, *LogicPro* prompts you to save your changes before it closes the Project.

Deleting a Project

NOTE! ***A Project must be closed before it will appear on the list of Projects that may be deleted.***

To delete a Project:

1. Select the **File** menu.
2. Select **Delete**.
3. Select **Project.** The **Delete Project** dialog box appears, as shown in *Figure 3.6 on page 32*.
4. Select the Project or Projects you want to delete from the list. By default, no projects are selected when the dialog box appears. Click on a Project once to select it for deletion. Click on it a second time to deselect it. Double clicking on an unselected Project is equivalent to selecting it and clicking the **[OK]** button.
5. Click **[OK]**. A message dialog box appears to confirm deletion of the Project and all its Resources and Programs.

NOTE! ***If you click* [OK]*, you remove the Project names from the list and delete all associated Resource and Program files. There is no way to recover this information. Be certain you want to delete these files before you click* [OK].**

6. Click **[OK]** to confirm the deletion
   – or –
   [Cancel] to abort the deletion.

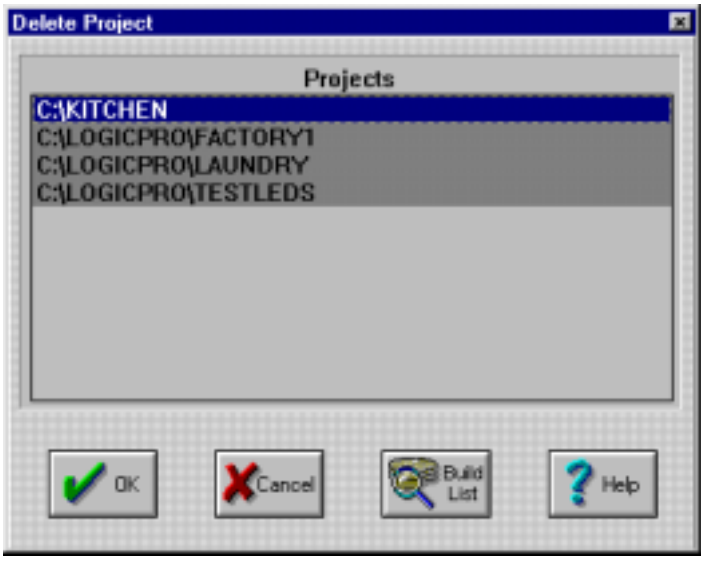


Figure 3.6 Delete Project Dialog Box

4

Resources

Resources are the individual controllers that execute the logic you created in a *LogicPro* Program.

Creating a Resource

To create a Resource:

1. Select the **File** menu.
2. Select **New**.
3. Select **Resource.** The **New Resource** dialog box appears, as shown in *Figure 4.1*.

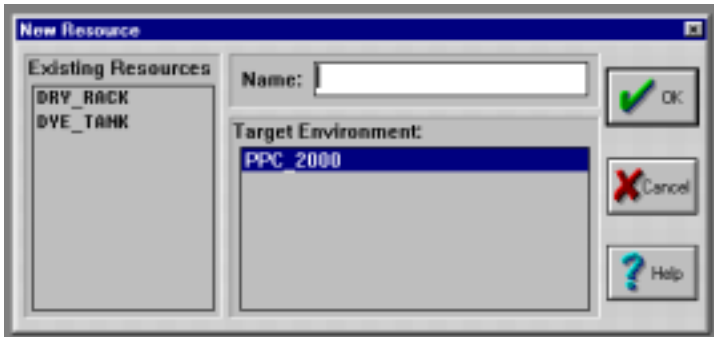


Figure 4.1 New Resource Dialog Box

4. Select **PPC_2000** or **CPC400** from the **Target Environment** list.

5. Enter a **Name** for the Resource you want to define. *LogicPro* saves this Resource as a database in the Project subdirectory.

6. Click **[OK]** to define the Resource. The **Attributes** dialog box appears.
   – or –
   Click **[Cancel]** to exit the dialog box.

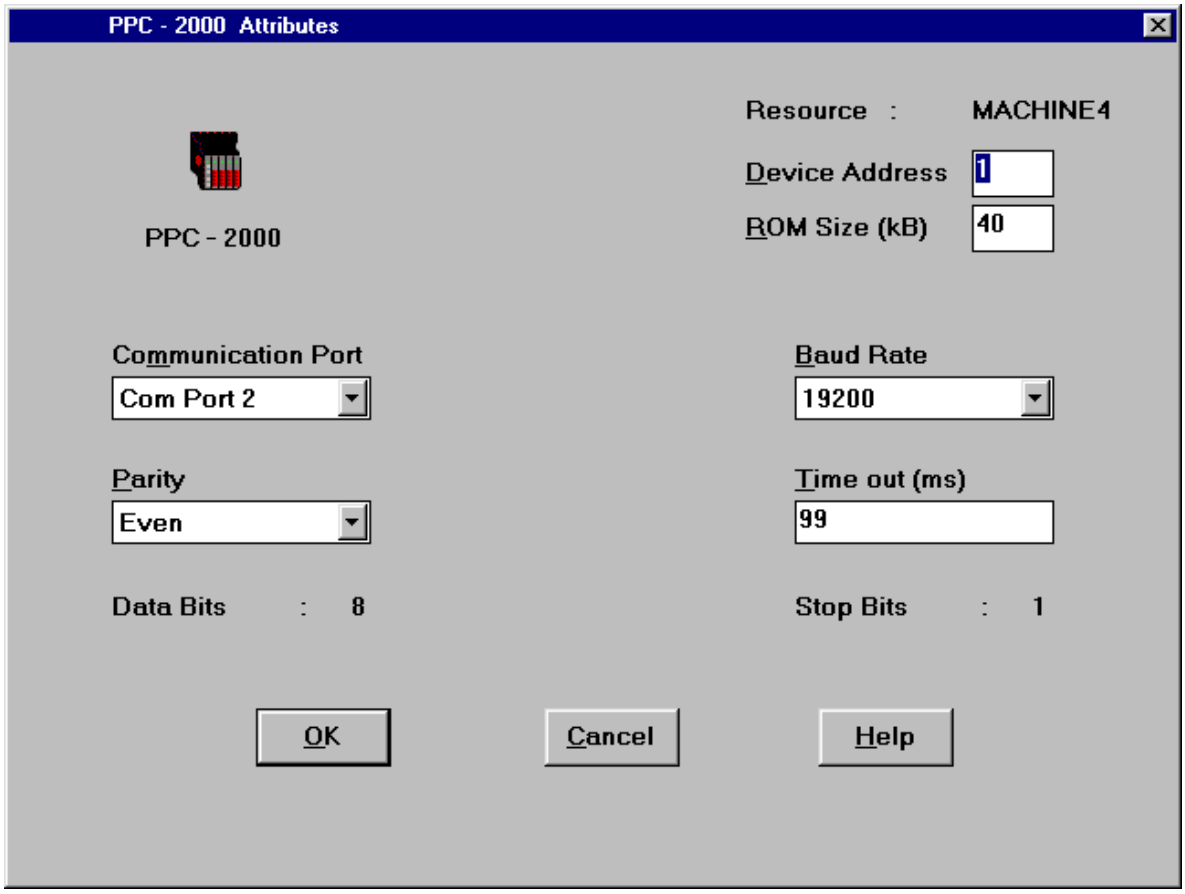


Figure 4.2 PPC-2000 Attributes Dialog Box

PPC-2000 Attributes

The following LogicPro default communications settings will match the PPC-2000's port 1 when the rotary CONFIG switch is set to position A.

- Baud: 19200
- Parity: Even
- Device Address: 1

To configure communications with the PPC-2000:

1. Select the correct **Communication Port** for your PC.
2. Select a **Parity** of **Even, Odd** or **None**.
3. In the **Device Address** field, enter the controller address for the PPC-2000.
4. In the **ROM Size** field, enter 128kB. (Controllers with 2.03 and earlier firmware support up to 40 kB.
5. Select a **Baud Rate**. The PPC-2000 supports **9600** and **19200**.
6. In the **Time out** field, enter 2000 milliseconds.
7. Click **[OK]**
   -or-
   Click **[Cancel]** to exit this screen.

CPC400 Attributes

The CPC400 has the following default communication settings:

- Baud Rate: 19200
- Controller Address: 1

To configure communications with the CPC400:

1. Select the correct **Communication Port** for your PC.
2. Select a **Parity** of **None**.
3. In the **Device Address** field enter the controller address you set in the CPC400.
4. In the **ROM Size** field, enter 64 kB.
5. Select a **Baud Rate**. The CPC400 supports **2400**, **9600** and **19200**.
6. In the **Time out** field enter 2000 milliseconds.
7. Click **[OK]**
   -or-
   Click **[Cancel]** to exit this screen.

Editing a Resource

To edit the open Resource:

1. Select the **Edit** menu.
2. Select **Attributes**.
3. Select **Resource**. The **Edit Resource** dialog box appears. See *Figure 4.3* with the current Resource type highlighted.
4. Click **[OK]** to open the **Attributes** dialog box
   – or –
   Click **[Cancel]** to exit the dialog box.
5. Make the desired changes to the settings.

NOTE! ***This is the same dialog box used when creating a* New Resource*. See "Creating a Resource" on page 33 for an explanation of the various Resource parameters.***

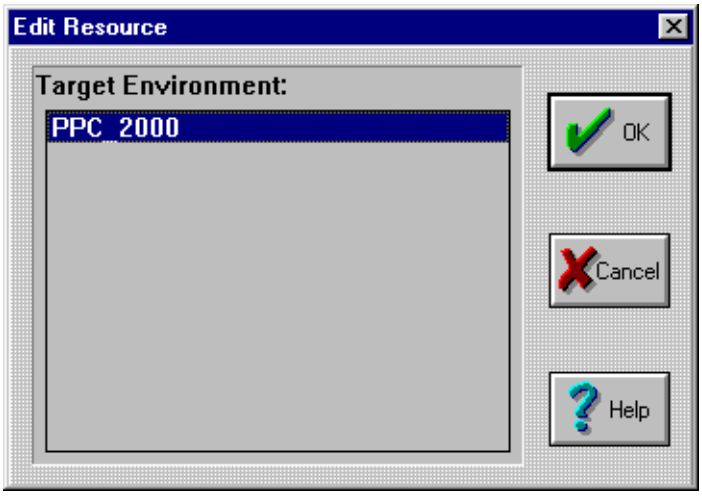


Figure 4.3 Edit Resource Dialog Box

Copying Resources

Resources must be copied within the context of their Projects. To copy a Resource:

1. Copy or back up the Project containing the target Resource and then open it. *See "Copying Projects" on page 28* and *See "Backing Up a Project" on page 29*

2. Since the project may contain many different Resources, delete any unnecessary Resources.

   a) Select **File**.

   b) Select **Delete**.

   c) Select **Resources**. The **Delete Resources** dialog box appears.

   d) Select the Resources to delete.

   e) Click **[OK]**.

The temp directories can be deleted to minimize the size of the project. Using any file utility such as Windows Explorer delete the temp directories (1bk~ through 5bk~) located within the Project directory.

Resource Backup

Backup, the precautionary copying of your Resource files to an alternative location, is an important step in protecting your work from equipment failure. You must meet two conditions in order to backup a Resource.

- The Resource must exist within the open Project.
- The Resource must be closed.

You cannot move or copy a Resource outside its original Project.

To backup a Resource:

1. Select the **File** menu.
2. Select **Backup**.
3. Select **Resource**. The **Copy Resource** dialog box appears, as shown in *Figure 4.4 on page 38*.
4. Select the Resource you want to back up by clicking on it in the **Existing Resources** list.
5. Enter a new name in the **New Resource Name** field.
6. Click **[OK]**.

NOTE! ***This is very useful for duplicating controller elements that repeat.***

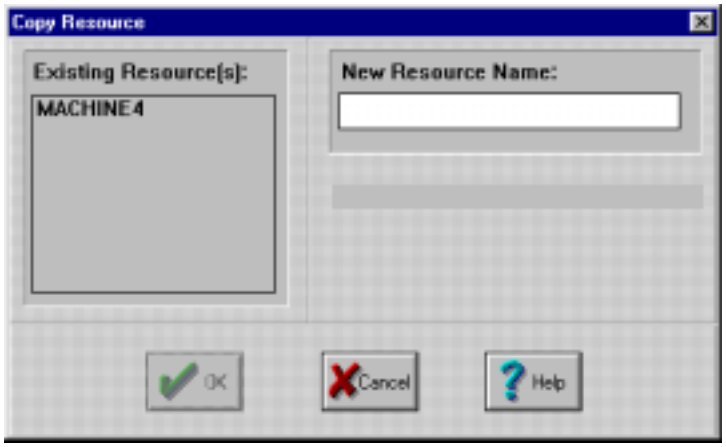


Figure 4.4 Copy Resource Dialog Box

Deleting a Resource

This option irreversibly removes an existing Resource from the current Project.

1. Select the **File** menu.
2. Select **Delete**.
3. Select **Resource**. The **Delete Resource** dialog box appears, as shown in *Figure 4.5 on page 39*.
4. Select the Resource or Resources you want to delete from the list. You can delete multiple Resources simultaneously. By default no Resources are selected when the dialog box appears. Click on a Resource once to select it. Click a second time to deselect it. Double clicking an unselected Resource is equivalent to selecting it and clicking **[OK]**.
5. Click **[OK]**. A message dialog box appears to confirm deletion of the Resource and all its Programs.

NOTE! ***By clicking* [OK]*, you remove the Resource name from the list and delete all associated Program files. There is no way to recover this information, so be absolutely certain you want to delete these files before you click* [OK].**

6. Click **[OK]** to accept the deletion
   – or –
   Click **[Cancel]** to abort the deletion.

Figure 4.5 Delete Resource Dialog Box

Opening a Resource

This option makes the Resource available to you.

1. Select the **File** menu.
2. Select **Open**.
3. Select **Resource**. The **Open Resource** dialog box appears, as shown in *Figure 4.6*.
4. Select the Resource you want to open from the list of Resources.
5. Click **[OK]**. The Resource opens displaying its name on the status line. You are then prompted to open a Program. *See "Opening a Program" on page 42*.

Figure 4.6 Open Resource Dialog Box

5

Programs

Programs form the base unit of the *LogicPro* development environment. Programs are defined in IEC 1131-3 as logical assemblies of all the programming language elements and constructs necessary for the intended signal processing needed to control a machine or process by a programmable controller system.

One or more Programs can be contained within a Resource. Within each Program, there may be any number (including zero) of steps, actions, relay elements, or function blocks.

While you can write multiple Programs for each controller, only one Program can actually run on the Resource at any time.

Creating a New Program

1. Select the **File** menu.
2. Select **New**.
3. Select **Program**. The **New Program** dialog box appears, as shown in *Figure 5.1 on page 42*.

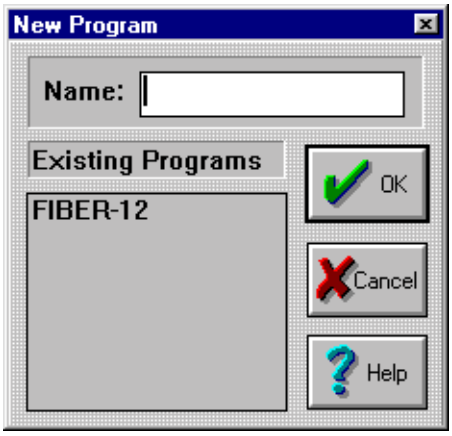


Figure 5.1 New Program Dialog Box

4. Enter the name for the new Program. Since the Program name is used as the name of DOS data files, the name must conform to DOS standards for file names. The name cannot be the same as the Resource name. See *Appendix A* for details on naming conventions.
5. Click **[OK]**.

Opening a Program

To open an existing Program:

1. Select the **File** menu.
2. Select **Open**.
3. Select **Program**. The **Open Program** dialog box appears, as shown in *Figure 5.2 on page 43*.
4. Select the Program you want to open from the list.
5. Click **[OK]**.

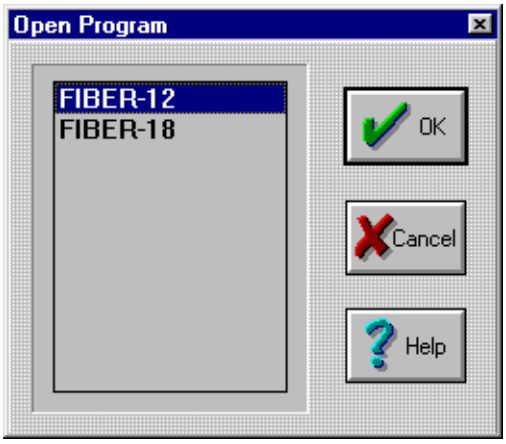


Figure 5.2 Open Program Dialog Box

Saving a Program

When you save a Program, *LogicPro* creates a backup file that contains the previously saved version of the Program. The Program appears as a `.THR` file and the backup file appears as a `.BAK` file in the Resource directory.

To save the currently active Program window:

1. Select the **File** menu.
2. Select **Save**.

Closing a Program

To close a Program:

1. Select the editor window with the Program you wish to close from the **Windows** menu if it is not already selected (on top).
2. Select the **File** menu.
3. Select **Close**.
4. Select **Program**. A message dialog box appears as shown in *Figure 5.3 on page 44* if any changes have been made and not saved.

5. Click **[Yes]** to save the Program and create a backup file of the old Program.
   – or –
   Click **[No]** to close the Program without saving.
   – or –
   Click **[Cancel]** to leave the Program open.

NOTE! ***Additions and deletions of variable definitions are saved when made and do not depend on saving the Program.***

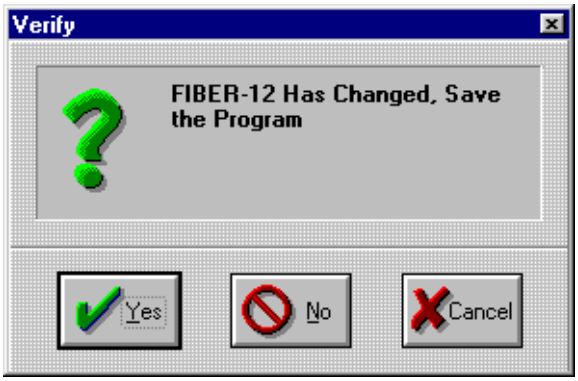


Figure 5.3 Verify Dialog Box

Copying Programs

Programs must be copied within the context of their Projects and resources. To copy a Program:

1. Copy or back up the Project containing the target Program and then open it. *"Copying Projects" on page 28.* and *"Backing Up a Project" on page 29.*
2. Since the Project may contain many Resources, delete all Resources that do not contain the target Program.
   a) Select **File**.
   b) Select **Delete**.
   c) Select **Resources**. The **Delete Resources** dialog box appears.
   d) Select the Resources to delete.
   e) Click **[OK]**.

3. Since the Resource may contain many different Programs, delete any unnecessary Programs.

   a) Select **File**.

   b) Select **Delete**.

   c) Select **Programs**. The **Delete Program** dialog box appears.

   d) Select the Programs to delete.

   e) Click **[OK]**.

The temp directories can be deleted to minimize the size of the Project. Using any file utility such as Windows Explorer delete the temp directories (1bk~ through 5bk~) located within the Project directory.

Saving a Program with a New Name (Save As)

The Save As feature allows you to save an open Program under a new name. To use this feature:

1. Select the **File** menu.
2. Select **Save As**.
3. Enter the new name you wish the Program to be saved as. A list of the existing Program names display below this field. *LogicPro* will not allow you to overwrite an existing Program.
4. Click **[OK]** to finish the save.
   – or –
   Click **[Cancel]** to go back to the Program window.

Backing Up a Program

You can backup or copy a Program if:

- The backed-up Program is located within the same Resource.
- The Program is closed.

You cannot move or copy a Program outside the original Resource.

To backup a Program:

1. Select the **File** menu.
2. Select **Backup**.
3. Select **Program**. The **Copy Program** dialog box appears, as shown in *Figure 5.4*.
4. Select the Program you want to backup by clicking on it in the **Existing Program(s)** list.
5. Enter the new Program name in the **New Program Name** field. A list of the existing Program names is displayed below this field to help insure that you do not use a name that is already assigned.
6. Click **[OK]** to finish the save
   – or –
   Click **[Cancel]** to abort the backup.

If a Program already exists in the Resource with the name you entered, you are prompted to confirm overwriting the existing Program.

7. Click **[OK]** to overwrite the existing Program
   – or –
   Click **[No]** to go back to the **Copy Program** dialog box and enter a unique name.

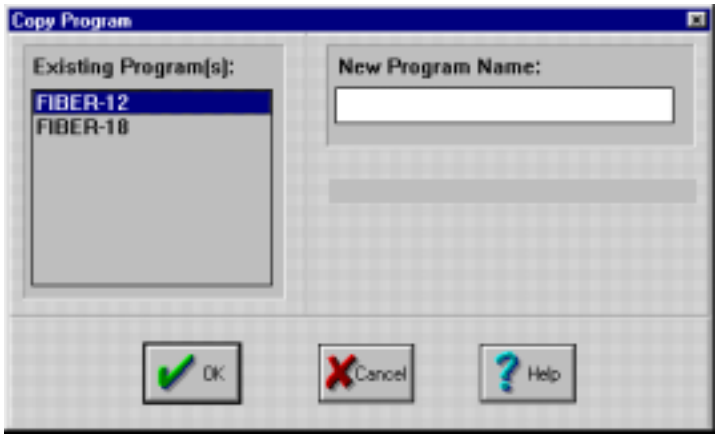


Figure 5.4 Copy Program Dialog Box

Deleting a Program

To delete an existing Program:

1. Select the **File menu**.
2. Select **Delete**.
3. Select **Program**. The **Delete Program** dialog box appears, as shown in *Figure 5.5*.
4. Select the Program you want to delete.

Multiple Programs can be selected by clicking on one Program name after another. To deselect a Program click on its name a second time.

5. Click **[OK]**. A message dialog box appears to confirm that you want to delete the Program and all its associated variables.

NOTE ! ***If you click* [OK]*, you remove the Program from the list and delete all associated files. There is no way to recover this information, so be certain you want to delete these files before you click* [OK].**

6. Click **[OK]** to delete the Program
   -or-
   [Cancel] to abort the deletion.

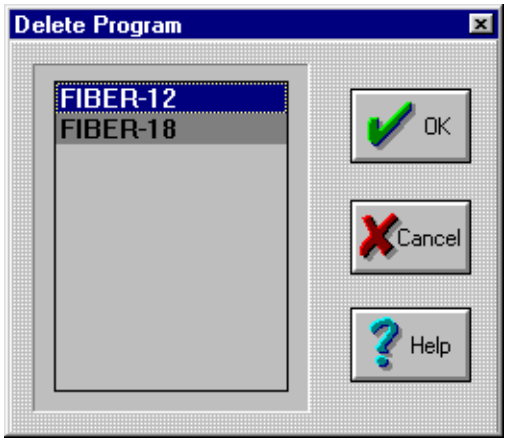


Figure 5.5 Delete Program Dialog Box

6

Variables

A *LogicPro* Variable is the name tag given to a piece of data used by various execution elements of the Program. Variables are both physical I/O and internally represented data. *LogicPro* uses several types of Variables, including:

- Project Variables
- Resource Variables
- Program Variables

Naming Conventions

When naming Variables try to be as descriptive as possible especially if there are more than a few Variables in the Program. For example, Go_Light_On will prove more useful a name than Variable_1.

Variables may be up to 28 characters long.

Initial Value—When a logic program is first run all of the variables are set equal to the initial values specified when the variables are declared or created. If an initial value is not explicitly entered in the **Add Variable** dialog box, the variable is set equal to zero. Unless a variable is set as retentive, it is set equal to its intitial value each time the logic program is run.

Retentive Variables

A Retentive Variable is a Variable that:

- Remembers its last state if power is interrupted.
- Returns that value after power has been restored.

LogicPro forgets all of the Variable values when a power down of a Resource occurs unless they are tagged as Retentive.

System Variables

System Variables give you access to internal system status and allow you to view items such as I/O Driver and Resource status. *LogicPro* generates these Variables automatically. You can access System Variables through the **Variable List** dialog box in the same manner as Project, Resource, and Program Variables.

Whenever an driver is referenced in your Program, *LogicPro* creates two Variables, Flag and Stat, for each driver.

- **Flag Variable** — is Boolean in nature and goes high if an error is detected.
- **Stat Variable** — is an integer and contains the river status bits or error code.

LogicPro also creates system variables whenever function blocks are added to a program. These variables can be used elsewhere in the program. The system variables are named by concatenating the user set name of the function block, and the function block's I/O symbols. For example, MYTIMER.Q is a system variable that holds a value of 0 when the output of function block named MYTIMER is off and a value of 1 when the output of that function block is on.

To view the existing system variables or assign one to a program element:

1. Double-click a program element.
2. For Variable Type, select **System**
3. For Data Type, select **All**

NOTE! ***You can view System Variables, only after having defined a Variable associated with a particular driver, and you cannot add, delete, or edit them.***

Project Variables

Project Variables are available to all elements in a Project. Project Variables are always internal, they cannot be a reference to physical I/O or database registers. To add, edit, and delete Project Variables:

1. Select the **Edit** menu.
2. Select **Variables**.
3. Select **Project**.

This will open the **Project Variables** dialog box, see *Figure 6.1 on page 51*.

4. To add a Variable, click **[Add]**. See *"Adding a Resource/Program Variable" on page 56.*
5. To clone a Variable, select it and click **[Clone]**. See *"Cloning a Project Variable" on page 55.*
6. To edit a Variable, select it and click **[Edit]**. See *"Editing a Project Variable" on page 53.*
7. To delete a Variable, select it and click **[Delete]**. See *"Deleting a Project Variable" on page 54.*
8. Click **[OK]** to accept the changes
   – or –
   Click **[Cancel]** to cancel all the changes.

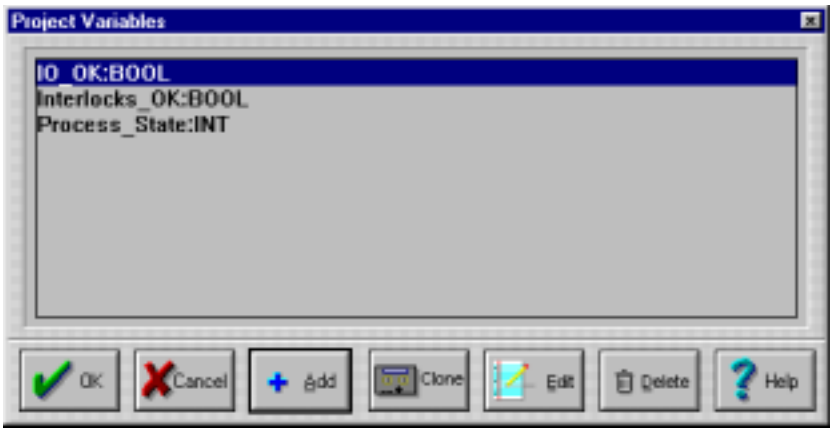


Figure 6.1 Project Variables Dialog Box

Adding a Project Variable

To add a Project Variable:

1. Click **[Add]** in the **Project Variable** dialog box. This dialog box to opens, as shown in *Figure 6.2*.

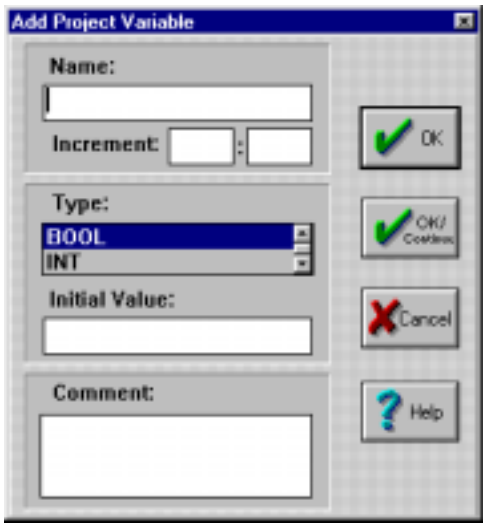


Figure 6.2 Add Project Variable

2. Enter the Variable name. This name must follow the Variable naming conventions discussed in *"Naming Conventions" on page 49*.
3. If you want to create several Variables with similar names, enter the root name in the **Name** field and enter a starting number and ending number in the **Increment** field. By this method, multiple Variables are created, each named with the root name and each appended with a number.
4. Select the Variable type from the list of types, either:
   - **BOOL** (Boolean)
   - **INT** (Integer)
   - **LONG** (Long integer)
   - **REAL** (Floating Point)
5. Enter the **Initial Variable Value**. If this field is left blank, zero is the default value. The Variable holds the value of zero until the Program changes that value. Initial value can be used for constant values of function blocks or constant true/false Variables.

Boolean Variables — are 1 bit in length and can have a value of 1 or 0, corresponding to true and false respectively.

Integer Variables — are 16-bit numeric Variables ranging in value from –32,768 to 32,767 including zero.

Long Integer Variables — are 32-bit (double-word) numeric variables ranging in value from -2,147,483,648 to 2,147,483,647.

Real Variables — are numeric variables with floating point decimals ranging in value from -3.4×10^{38} to 3.4×10^{38}.

6. Enter any comments you wish to make about the Variable in the **Comment** field.

7. Click **[OK]** when all the parameters for this Variable are defined.
   – or –
   Click **[OK/Continue]** to add the Variable and keep the **Add Project Variable** dialog box open and add another Variable.

The Variable with the parameters you selected now appears in the dialog box list. The next time you add a Variable, the **Add Variable** dialog box remembers the previously entered type and size fields. This makes it easier and faster to add many similar Variables to a Project.

Editing a Project Variable

To edit a Project Variable:

1. Highlight the Variable you want to edit in the list in the **Project Variable** dialog box.

2. Click **[Edit]**. The **Edit Project Variable** dialog box appears, as shown in *Figure 6.3 on page 54*.

3. Edit the Variable name, type or initial value.

4. Click **[OK]** when all the parameters for this Variable are defined.

With the **Edit Project Variable** dialog box open you can:

- Open a new **Edit Project Variable** dialog box and add a New Variable by clicking the **[Add]** button.
- Duplicate the existing Variable's settings in a new **Edit Project Variable** dialog box by clicking the **[Clone]** button.
- Delete the Variable you are editing by clicking the **[Delete]** button.

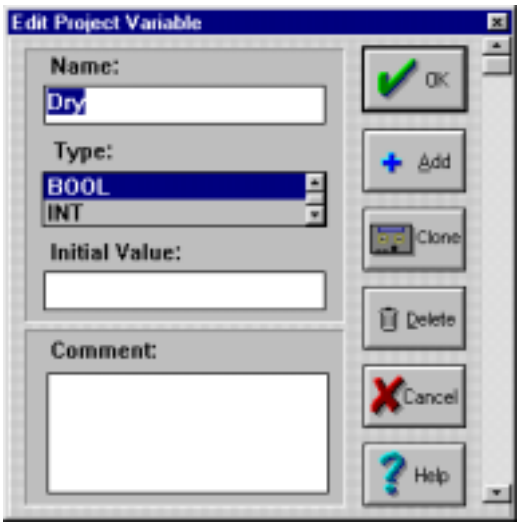


Figure 6.3 Edit Project Variable Dialog Box

Deleting a Project Variable

To delete a single Project Variable:

1. Select the Variable you want to delete in the **Project Variables** dialog box.
2. Click the **[Delete]** button. The **Verify** dialog box appears.

NOTE! ***You are prompted to confirm deletion only if the Options/Variable/Confirm/Deletion choice is selected.***

3. Click **[Yes]** to delete the Variable
   – or –
   Click **[No]** to cancel the deletion.

To delete multiple Variables simultaneously:

1. Select the first Variable you want to delete in the **Project Variables** dialog box.
2. To select another non-sequential Variable:
   a) Press and hold **<CTRL>**.
   b) Click the additional Variables you want to delete. Only the first selection and the additional Variables are highlighted.

To select a sequence of Variables:

c) Press and hold **<SHIFT>**.

d) Click on the last in the sequence of Variables you want to delete. This action highlights the first selection, the last selection, and all of the Variables in between the two.

3. Click **[Delete]**. The **Verify** dialog box appears.

NOTE! ***You are prompted to confirm deletion only if the Options/Variable/Confirm/Deletion choice is selected.***

4. Click **[Yes]** to delete the Variables.
   –or –
   Click **[No]** to cancel the deletion.

NOTE ! ***If you make a mistake, you can use the* [Cancel] *button to restore all settings to the state in which they existed when you first opened the Project Variables dialog box.***

Cloning a Project Variable

Cloning a Variable creates an exact duplicate of that Variable and highlights the symbolic name of the copied Variable. This makes the creation of similar Variables easier.

To clone a Project Variable:

1. Select the Variable you want to clone from the list in the **Project Variables** dialog box.
2. Click **[Clone]**. The **Add Project Variable** dialog box is opened with the cloned Variable.
3. Edit the Variable name. Give the Variable a unique name.
4. Edit any other parameters that you wish to be different from the model for the cloned Variable.
5. Click **[OK]** to add the Variable
   – or –
   Click **[Cancel]** to return to the **Project Variable** dialog box without adding the cloned Variable.
   –or –
   Click **[OK/Continue]**. This button behaves like the standard **[OK]**. The only difference between them is that **[OK/Continue]** will not cause the dialog box to close.

However, just like the **[OK]**, all of the information that you entered or changed is confirmed.

NOTE ! ***If you make a mistake, you can use the* [Cancel] *button to restore all settings to the state in which they existed when you first opened the Project Variables dialog box.***

Resource Variables

A Resource Variable can be internal or can be associated with a physical I/O point or a database register. Resource Variables are available to all Programs written for a Resource. To add, edit, and delete Resource Variables:

1. Select the **Edit** menu.
2. Select **Variables.**
3. Select **Resource**. The **Resource Variables** dialog box appears.
4. Add Variables by clicking **[Add]**, edit Variables by clicking **[Edit]**, or delete Variables by clicking **[Delete]**. Additional information on these functions is available in the following pages.
5. Click **[OK]** to accept the changes
   – or –
   Click **[Cancel]** to cancel all the changes you made to Variables.

Adding a Resource/Program Variable

Resource and Program Variables are added in the same manner. To add a Resource/Program Variable:

1. Select the **Edit** menu.
2. Select **Variables**.
3. Select **Resource** or **Program**. This opens the **Resource Variables** or **Program Variables** dialog box, as shown in *Figure 6.4* and *Figure 6.5 on page 57*.

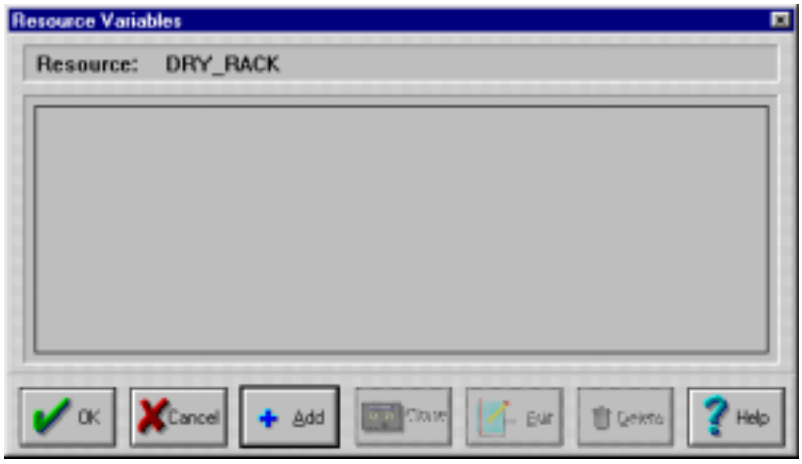


Figure 6.4 Resource Variable Dialog Box

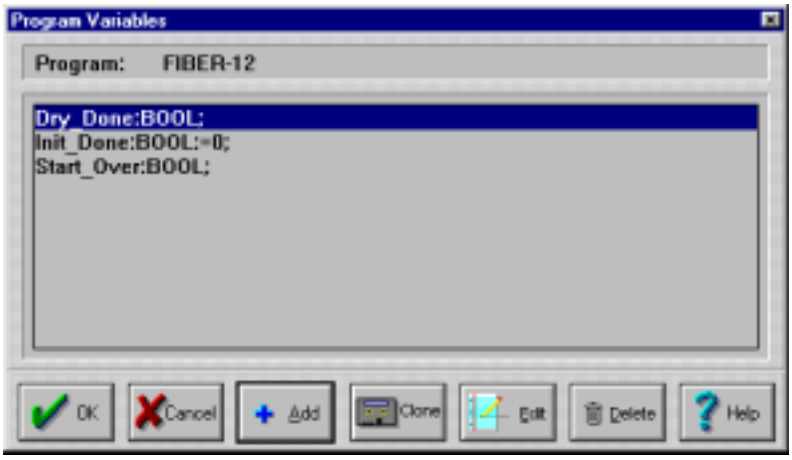


Figure 6.5 Program Variable Dialog Box

4. Click **[Add]** in the **Resource/Program Variables** dialog box. The **Add Resource/Program Variable** dialog box appears, as shown in *Figure 6.6 on page 58*.

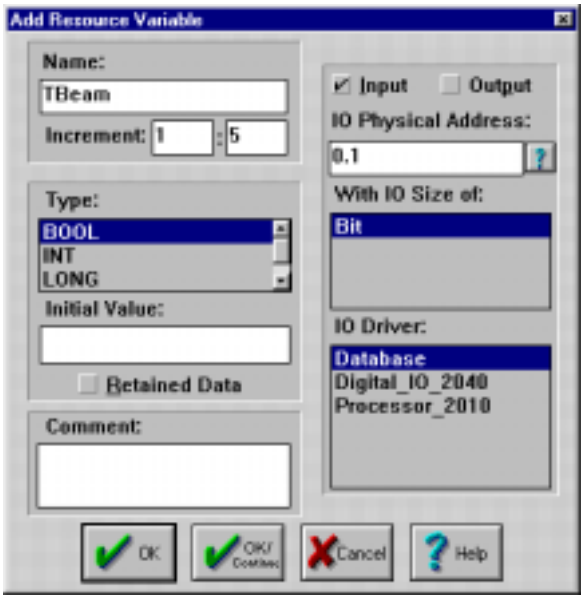


Figure 6.6 Add Resource Variable Dialog Box

5. Enter the Variable name in the **Name** field. This name must follow the Variable naming conventions discussed in *"Naming Conventions" on page 49*.

6. If you want to create a numbered range of Variables, enter a start and an end number in each of the **Increment** fields. Otherwise, leave the fields blank.

7. Select the Variable **Type** from the list of types, either:

   - **BOOL** (Boolean)
   - **INT** (Integer)
   - **LONG** (Long Integer)
   - **REAL** (Floating Point)

Refer to *"Adding a Project Variable" on page 52* , for a more detailed discussion of allowable parameters for Variable types.

8. Enter the **Initial Variable Value**. If this field is left blank, zero is the default value. The Variable holds the value of zero until the Program or an I/O reading changes that value. Initial value can be used for constant values of function blocks or constant true/false Variables.

9. To associate a Variable with a physical I/O point or a database register, follow these steps:

a) Place a check mark in either the **Input** or the **Output** check box.

b) Select the I/O driver from the list. The drivers in the list are determined by the Resource you selected. Information on these drivers is located in *Chapter 12, "I/O Drivers."*

c) Enter the I/O physical address. See *Chapter 12, "I/O Drivers,"* for addressing information.

d) Select one of the following I/O sizes from the list of sizes:

- **Bit** — a binary digit that can hold the value of 0 or 1.
- **Word** — 16 bits in length.

10. Enter any comments in the **Comment** field.

11. Click **[OK]**.

NOTE! ***If the I/O point is an Input, you must choose Input. If the point is an output, choose* Output *if you intend for the logic Program to set the output's state. You may also choose* Input *for a physical output if its state is set by the closed-loop control Program and the logic Program is only interested in sensing the value or state of the physical output. For database registers, choose* Input *or* Output *based on whether the logic Program sets the value (output) or just reads it (input).***

The Variable appears in the dialog box list. The next time you add a Variable, the **Add Variable** dialog box remembers the previously entered type and size fields. This makes it easier and faster to add many similar Variables to a Resource/Program.

Editing a Resource/Program Variable

Resource and Program Variables are edited in the same manner. To edit a Resource/Program Variable:

1. Select the Variable you want to edit in the **Resource/Program Variables** dialog box.

2. Click **[Edit]**. The **Edit Variable** dialog box appears.

3. Enter the appropriate information in all the dialog box fields, according to the procedure in *"Adding a Resource/Program Variable" on page 56*.

Deleting Resource/Program Variables

To delete a single Resource/Program Variable:

1. Select the Variable you want to delete in the **Resource/ Program Variables** dialog box.
2. Click **[Delete]**. The **Verify** dialog box appears.

NOTE! ***You are prompted to confirm deletion only if the Options/Variable/Confirm/Deletion choice is selected.***

3. Click **[Yes]** to delete the Variable
   – or –
   Click **[No]** to cancel the deletion.

To delete multiple Variables simultaneously:

1. Select the first Variable you want to delete in the **Resource/Program Variables** dialog box.
2. To select another non-sequential Variable:
   a) Press and hold **<CTRL>**.
   b) Click the additional Variable you want to delete. Only the first selection and the additional Variables are highlighted.

To select a sequence of Variables:

   a) Press and hold **<SHIFT>**.
   b) Click on the last of the Variables in the sequence that you want to delete. This action highlights the first selection, the last selection, and all of the Variables in between the two.

3. Click **[Delete]**. The **Verify** dialog box appears.

NOTE! ***You are prompted to confirm deletion only if the Options/Variable/Confirm/Deletion choice is selected.***

4. Click **[Yes]** to delete the Variables.
   – or –
   Click **[No]** to cancel the deletion.

NOTE! ***If you make a mistake, you can choose* [Cancel], *in the Resource/Program Variables dialog box, to restore all Variables to the state in which they existed when you first opened the Resource/Program Variables dialog box.***

Cloning a Resource or Program Variable

Cloning a Variable creates an exact duplicate of that Variable and highlights the symbolic name of the copied Variable. This makes the creation of similar Variables easier.

To clone a Resource Variable:

1. Select the Variable you want to clone from the list in the **Resource Variables** or **Program Variables** dialog box.
2. Click **[Clone]**. The **Add Resource/Program Variable** dialog box is opened with the cloned Variable.
3. Edit the Variable name. Give the Variable a unique name.
4. Edit any other parameters that you wish to be different than the model for the cloned Variable.
5. Click **[OK]** to add the Variable.
   – or –
   Click **[Cancel]** to return to the **Resource/Program Variables** dialog box without adding the cloned Variable.

Program Variables

Program Variables are available to all elements in a Program. A Program Variable can be internal, or it can be associated with a physical I/O point or a database register. To add, edit, and delete Program Variables:

1. Select the **Edit** menu.
2. Select **Variables**.
3. Select **Program**.
4. Add Variables by clicking **[Add]**. See *"Adding a Resource/Program Variable" on page 56*.
   -or-
   Clone Variables by clicking **[Clone]**. See *"Cloning a Resource or Program Variable" on page 61*.
   -or-

Edit Variables by clicking **[Edit]**. See *"Editing a Resource/Program Variable" on page 59.*
-or-
Delete Variables by clicking **[Delete]**. See *"Deleting Resource/Program Variables" on page 60.*

5. Click **[OK]** to accept the changes.
   – or –
   Click **[Cancel]** to cancel all the changes you made to the Variables.

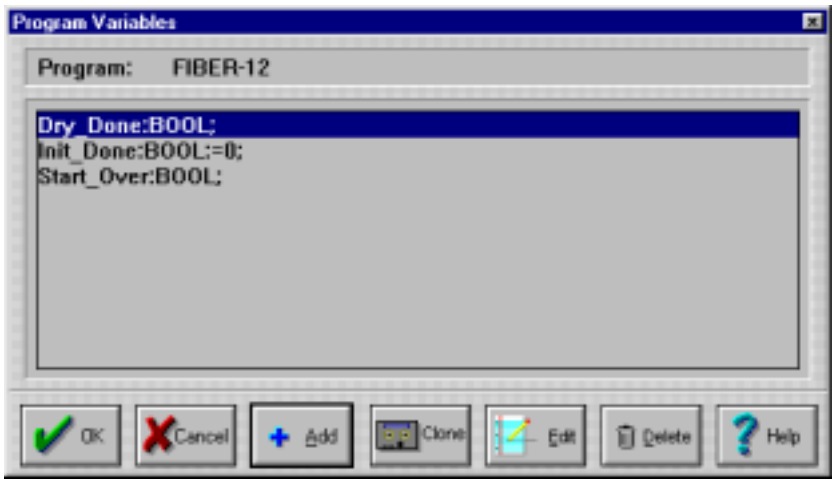


Figure 6.7 Program Variables Dialog Box

Finding and Replacing a Variable In a Program

You can use the **Find** or **Replace** commands to help you edit ladder diagrams.

Using the Find Command

You can use the **Find** command two ways:

- To locate a particular Variable name, or
- To locate subsequent instances of a Variable name.

To use the **Find** command:

1. From the **Edit** menu.
2. Select **Find**. The **Find Variable** dialog box appears, as shown in *Figure 6.8 on page 63*.
3. Enter the name of the Variable you want to find in the **Variable to find** field.

4. Select whether you want to start your search at the beginning of the Program at your current location.

5. Click **[OK]** to find the first instance of the Variable name.

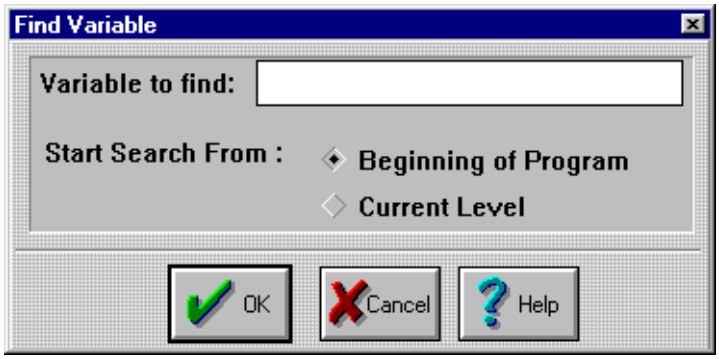


Figure 6.8 Find Variable

If you click...	Then...
[OK]	The first instance of the Variable name is highlighted in the Ladder Diagram. The **Find Variable** dialog box closes. A message dialog box appears.

6. To find additional instances of the Variable name, click **[Yes]** in the message dialog box.
   – or –
   Click **[No]** to end the search.

When...	Then...
There are no additional instances of the Variable name.	The dialog boxes closes and the editor window becomes active.
The Variable name does not exist in the Ladder Diagram	A warning box appears with the message **Variable not found!** Click **[OK]** to clear the dialog box from the screen.

Using the Replace Command

When...	Then...
There are no additional instances of the Variable name.	The dialog boxes closes and the editor window becomes active.
The Variable name does not exist in the Ladder Diagram	A warning box appears with the message **Variable not found!** Click **[OK]** to clear the dialog box from the screen.

You can use the Replace command two ways:

- To locate and replace the first instance of a particular Variable name, or
- To locate and replace subsequent instances of a Variable name.

To use the Replace command:

1. Select the **Edit** menu**.**
2. Select **Replace.** The **Replace Variable(s)** dialog box appears.
3. Enter the name of the Variable you want to find in the **Variable to find** field.
4. Enter the name of the new Variable in the **New Variable** field. This new name will replace the old Variable name.
5. Click **[OK]** to find the first instance of the Variable name.

When...	Then...
The Variable is found.	The first instance of the Variable name is highlighted. A message dialog box appears asking you to confirm replacing the instance of the found Variable.
There are no additional instances of the Variable name.	The dialog boxes closes and the editor window becomes active.
The Variable name does not exist in the Ladder Diagram	A warning box appears with the message **Variable not found!** Click **[OK]** to clear the dialog box from the screen.

6. Click **[Yes]** to replace the found instance of the Variable and find the next instance, repeat step #5.
   – or –
   Click **[No]** to find the next instance without changing the found instance, repeat step #5.
   – or –
   Click **[Cancel]** to stop searching without replacing the found instance.

Import/Export

Import/Export is a feature that allows the user to import (load) or export (save) Resource or Program Variables to or from an ASCII file. You can create or edit these files outside of *LogicPro*.

Importing an ASCII file

Importing is the process of loading an ASCII file containing Variable information into a *LogicPro* Resource or Program.

To import an existing ASCII file, use the following procedure:

1. Select the **File** menu.
2. Select **Import Variable**.
   a) Select **Project**
   -or-
   Resource
   -or-
   Program. The **Import Variable** dialog box appears.
3. Select the type of import operation you want to perform:
   - **Append**
   - **Replace All**
   - **Replace I/O Variables Only** (Resource and Program only)

Append

Selecting **Append** adds the contents of the ASCII file to the existing Project, Resource, or Program Variables.

If a Variable from the import file has the same name as an existing Variable in your *LogicPro* Resource or Program, a **Verify** dialog box appears prompting you to replace the Variable.

If you select...	Then...
Yes	The new Variable data in the ASCII file replaces the Variable data presently in the Resource or Program file.
No	The new Variable data in the ASCII file is ignored and the existing Resource or Program data is unchanged.

Replace All

Selecting **Replace All** replaces all the existing Resource or Program Variables with corresponding Variables in the import file, with the following qualifications:

- If a new Variable name from the ASCII file is identical to an existing Variable in your Resource or Program Variables, the existing Variable is replaced by the new one.
- If a new Variable name from the ASCII file is not identical to any existing Resource or Program Variable name, a **Verify** dialog box opens informing you that the specific Variable does not exist. The dialog then prompts you to Append the Variable.

If you select...	Then...
Yes	The Variable is added to your Resource of Program Variable.
No	The new Variable is ignored.

- If a Variable exists in your Resource or Program and no duplicate named Variable exists in the import file, it is unaffected and still exists after the import operation is complete.

Replace I/O Variables Only

Replace I/O Variables only results in a similar effect to the **Replace All** option except that only the I/O Variables are affected as described above. Any other Variables are not affected in any way.

Verifying the Import Operation

After importing a file, you can verify the result by opening the Resource or Program Variables within *LogicPro* and observing the Variable list. You can edit each individual Variable to ensure that the **Import** operation executed correctly and the result is what you expected.

Creating or Editing an ASCII file

You can create a file for import using any ASCII text editor, such as Microsoft's MS-DOS Editor. Word processors like Microsoft's Word, are not ASCII text editors and they may add control and formatting codes that make the file unusable in *LogicPro.* If you want to use a Word Processor for this function you must save the file as a text file. Consult your Word Processor manual for more information.

The file you want to import must have a *.TXT extension and use a comma-delimited field format.

File Format Specification of Import/Export

The content of the file is defined as follows:

The first line — indicates where the file will be imported to or exported from. The legal form of the first line is either **[PROJECT:ProjName], [RESOURCE:ResName]** or **[PROGRAM:ProgName]**, where PROJECT, RESOURCE, and PROGRAM indicate the type of Variables to be imported or exported and ProjName, ResName, and ProgName are names for the Project, Resource, and Program.

The 2nd line — should be empty [ ]. It is reserved for future use.

The last line — must be [END].

The other lines have their data separated by commas and each line corresponds to one Variable to be imported or exported. Each of the items are defined in the *Table 6.1 on page 68*.

Table 6.1 File Format Specification of Import/ Export

Item Name	Max Size	Legal Values	Comments
Variable Name	26	1st char must be letter. Others must be alpha-numeric or '_'.	
Variable Type Short	4	BOOL, INT, LONG, REAL	BOOL for Boolean, INT for Integer, LONG for Long Integer, REAL for Real Type. See *Note 4*.
Retentive Flag	1	0, 1	1 indicates a Retentive Variable.
Initial Value	10	Digits only	See *Note 5*.
Comment	26	ASCII chars	
Input or Output	1	I, Q	I for input, Q for Output. See *Note 4*.
Size	1	X, B, W	X for bit, B for byte, W for Word. See *Note 4*. The size indicates how I/O is mapped to a Variable.
Driver Name	26	1st char must be letter. Others must be alpha-numeric or '_'.	See *Note 4*.
Address	49	digits.digits	This item must be in the form of moduleaddress.offset or parameternumber.offset as required by the driver; only digits and the decimal are allowed, and these digits must also be in the legal range specified by the particular drivers.

1. If the Variable type is defined as real, then Input/Output, Size, Driver Name and Address are not be available.
2. The Variable Name, Variable Type Short, Retentive Flag must be present for all Variables. The Initial Value is optional and defaults to zero. The **Comment** field may be left empty.
3. If the Variable is an I/O Variable, the input/output, Size, Driver name, Address must also be present.

4. Variable Type Short, Input or Output, Size and Driver Name are case sensitive.

5. Initial Values: See *"Naming Conventions" on page 49*.

Example of an Import/Export File

```
[RESOURCE:PPC_2000]
[]
Alarm_Status1,INT,0,,,I,W,Database,95.1
Analog_In1_1,INT,0,,,I,W,Analog_In_202x,1.1
Analog_Out11_1,INT,0,,,Q,W,Encoder_Analog_2030,11.1
Control_Mode_I1,INT,0,,,I,W,Database,1.1
Control_Mode_O1,INT,0,,,Q,W,Database,1.48
Dig_In1,BOOL,0,,,I,X,Processor_2010,0.1
Dig_Out25,BOOL,0,,,Q,X,Processor_2010,0.25
Encoder_In11_1,INT,0,,,I,W,Encoder_Analog_2030,11.1
Heat_Output_I1,INT,0,,,I,W,Database,36.1
Heat_Output_O1,INT,0,,,Q,W,Database,36.1
Setpoint_I1,INT,0,,,I,W,Database,95.1
Setpoint_O1,INT,0,,,Q,W,Database,95.1
[END]
```

Table 6.2 Selected Explanations of the Example Import/Export File

Code	Explanation
`[RESOURCE:PPC_2000]`	The first line indicates that these are RESOURCE type Variable and the name of the RESOURCE is PPC_2000.
`[ ]`	The second line is empty except for the opening bracket ([ ) and closing bracket ( ]).
`Alarm_Status1,INT,0,,,I,W,Database,95.1`	Variable Name, Variable Type Short, Retentive Flag, Initial Value, Comment, Input or Output, IO Size, Driver Name, IO Address.
`Analog_In1_1,INT,0,,,I,W,Analog_In_202x,1.1`	Variable Name, Variable Type Short, Retentive Flag, Initial Value, Comment, Input or Output, IO Size, Driver Name, IO Address.

`Dig_Out25,Bool,0,,,Q,X,Processor_2010,0.25`	Variable Name, Variable Type Short, Retentive Flag, Initial Value, Comment, Input or Output, IO Size, Driver Name, IO Address.
[END]	This is the last line in the ASCII file.

Exporting an ASCII file

The procedures for exporting Resource or Program Variables to an ASCII file are similar to the procedures for importing and ASCII file. When you export, you create or overwrite an ASCII file with the Resource or Program Variable information already in *LogicPro*.

To export Resource or Program Variables to an ASCII file, use the following procedure:

1. Select the **File** menu.
2. Select **Export Variable**.

   a) Select **Project**
   -or- **Resource**
   -or- **Program**
   The **Export Variable** dialog box appears.

3. Select the type of export operation you want to perform:

   Export All — exports all types of Variables.

   Export I/O Variables Only — exports all Variables defined as Input or Output, excluding all internal or non-IO defined Variables from the Export file (available for Resource and Program Variables.)

Verifying the Export Operation

After exporting a file, you can verity the result by using an ASCII based editor to view the file. Be sure to use an ASCII based editor or the file may have control and formatting codes added to it that will make it unsuitable for future import operations.

7

Sequential Function Charts

Sequential Function Chart (SFC) is a graphical language using simple graphical objects to diagram any sequential process. SFC enhances communication between the project manager, system architect, programmers, and operators by providing a clear graphical representation of the process state machine.

About SFC

SFCs provide a way to partition a controller program into a manageable set of Steps and Transitions. These Steps and Transitions are interconnected by directed links.

Steps include related Actions (such as filling a tank, counting parts or waiting for a delay) that are carried out until a Transition condition is satisfied. Once the Transition condition is satisfied, control is transferred to the subsequent step.

Using SFCs in *LogicPro*

SFCs support both serial and parallel sequential processes. Because discrete manufacturing and process applications all run as a sequence of Steps, SFCs are an excellent way to logically assemble the sub-processes into a structured program. Using these basic elements:

- Steps
- Transitions
- Actions

SFCs allow you to design, program, debug, and document your process in a quick and effective manner.

SFC is the default language for *LogicPro* programs.

When SFCs are the active language for the window, *LogicPro*:

- Displays the SFC toolbar on the right side of the screen.
- Displays the SFC tools on the **Tools** menu.

SFC Toolbar

SFC uses graphic elements to represent Steps, Transitions, and Actions in the program. In *LogicPro*, you can access these objects with the toolbar at the right of the screen.

You can activate any of the SFC tools by clicking on it on the toolbar. The selected tool remains active until a different tool is chosen.

Table 7.1 shows the SFC tools with a brief description of each tool's function. The following sections describe the tools and their use in programs in greater detail.

Table 7.1 The SFC Toolbar

This Tool	Performs This Function
	Selector
	Origin Step
	Step
	Simultaneous Transition (Directed Link)
	Loop Back (Connection)
	Action
	Comments
	Cross Reference

Selector

You can use the **[Selector]** tool to choose the elements in the program you want to move, edit, delete, or to which to assign variable names. When you click on any element in the program, handles highlight the object to show it is selected.

To select a single element, click on the object with the mouse. You can now move, delete or edit the selected object by using the appropriate tool on the tool bar.

To deselect an element click on another element or a blank space in the SFC editor window.

NOTE! ***Right-clicking while using any other tool reactivates the Selector tool.***

Cut, Copy, and Paste

You can cut, copy, and paste SFC Actions into other SFC programs or Actions. To select an Action, use the selector tool to get the handles. When pasting an Action you are prompted for a unique name for the Action if pasting would cause ambiguity.

Origin Step

All programs require an origin Step to begin the process program. Each SFC program can have one or two Origin Steps. Origin Step is initially active when the program is run.

The **[Origin Step]** tool allows you to insert an origin Step into the program window. Insert your Origin Step by selecting the **[Origin Step]** tool and clicking where you want the Step positioned.

Step

Steps are related Actions (such as filling a tank, counting parts or waiting for a delay) that are carried out until a Transition condition is met. When scanning the active states, Steps are equivalent to the logic associated with the states. Steps are either active or inactive. Actions associated with a Step are only active when the Step is active. At a given moment, the state of the program is defined by its active Steps and the values of their inputs and output variables.

You can insert Steps into your SFCs with the **[Step]** tool. Select the **[Step]** tool from the toolbar, click anywhere on the screen, and place the Step into the application work area as needed.

Simultaneous Transition

Transitions determine when the process flow passes from a preceding Step to the successive Step along the directed link. Simultaneous Transitions are represented by a horizontal line across the vertical directed link. Each Transition has an associated Transition condition, a single Boolean expression. Transition names are local to the program in which the Transition is located.

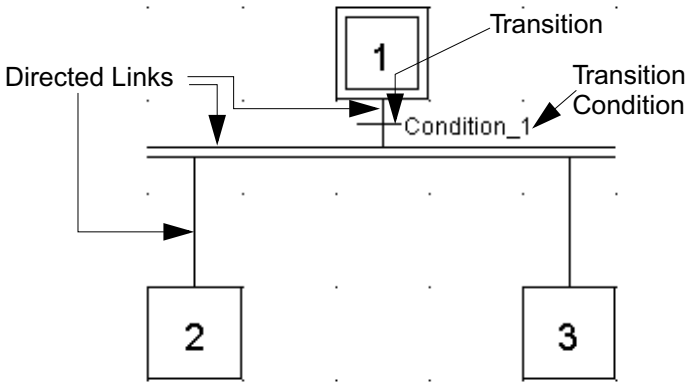


Figure 7.1 Simultaneous Transition

Inserting a Simultaneous Transition

To insert a Transition from one Step to the next:

1. Select the **[Simultaneous Transition]** tool.
2. Click and drag from the inside of one Step to the inside of another Step. A directed link with one Transition appears.
3. To add parallel directed links, click and drag from the Transition to the inside of a Step.

Parallel (AND) and Single (OR) Transitions

To toggle between parallel and single Transitions, using the **[Simultaneous Transition]** tool, double-click near the directed link on the opposite side of the double horizontal line from the Transition.

Parallel horizontal lines indicate that steps occur in parallel. For example, in the SFC as illustrated in *Figure 7.2*, both Steps 2 and 3 will execute in parallel once condition 1 is met.

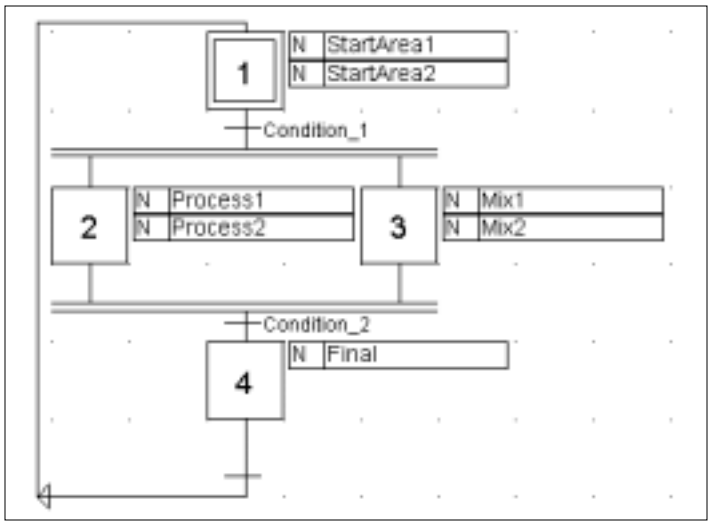


Figure 7.2 Parallel and / or Single Transitions

A single horizontal line indicates that only one step executes. For example, in *Figure 7.3*, either Step 2 or Step 3 will become active depending on whether condition 1 or condition 2 becomes true. In case both Transition conditions become true at the same time, Step 2 will become active because logic is executed top to bottom, left to right.

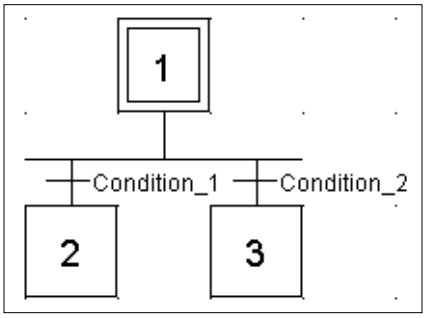


Figure 7.3 Simultaneous Transition Conditions

Assigning Variables to Transitions

Transition conditions must take the form of a single expression. A Transition condition can be a variable that allows the program to continue once the variable value is greater than 0. It could also be one or more variables related by Boolean operators resulting in a Boolean true or false result. Any integer value greater than 0 is considered true.

The following list of operators, shown in *Table 7.2*, may be used in the Transition condition expression:

Table 7.2 Transition Condition Expressions

Operator Symbol	Function
*	Times
/	Divided By
+	Plus
-	Minus
>=	Greater Than or Equal To
>	Greater Than
<=	Less Than or Equal To
<	Less Than
==	Equals
!=	Not Equal To
!	Not
&&	And
||	Or
&&!	And Not
||!	Or not
(	Left Parenthesis
)	Right Parentheses

To assign a logical condition to a Transition:

1. Select the **[Selector]** tool.
2. Double-click on the Transition to which you want to assign a condition. The **Transition Condition** dialog box appears.
3. Select a variable for the transition by selecting the appropriate checkbox and double-clicking the variable name in the list.
4. If the condition expression needs an operator, press the **Operator List** button on the dialog box. Add the operator by double-clicking it.

5. Select the appropriate checkbox in the **Variable Type** section of the dialog box and click **[Add], [Edit], [Delete],** or **[Cancel]** to define any variables you need. The appropriate dialog box appears and allows you to define the variables. Once a variable is defined, it appears in the **Variable List** dialog box. See *"Project Variables" on page 51* and *"Resource Variables" on page 56* for more details adding, editing, deleting and cloning variables.
6. Repeat steps 3 to 5 as needed to complete the transition condition expression.
7. Click **[OK]**. The Transition condition now appears next to the Transition in the program.

This condition can be changed by double-clicking on the Transition and constructing another condition.

NOTE! ***All Transitions in a program must have a Transition condition or you will encounter a code generator error.***

Loop Back

The **[Loop Back]** tool lets you insert a Transition loop back into the program window, where control is passed back to a previous Step. Loop backs follow the same rules as Transitions. Refer to *"Simultaneous Transition" on page 74* for complete information on defining Transition variables.

To insert a Transition loop back:

1. Select the **[Loop Back]** tool.
2. Click and drag the cursor from the center of one Step to the center of another Step higher in the control sequence.

Action

The **[Action Tool]** allows you to associate Actions with a particular Step in the program window.

You can associate zero or more Actions with each Step. A Step with zero associated Actions is considered a Wait function. The Step is waiting for a successor Transition condition to become true. An Action can be a collection of rungs in Ladder language, a Sequential Function Chart or a Function Block Diagram. The scope of the declaration of an Action is local to the program organization unit containing the declaration.

Like variables, Action names can be any combination of letters and digits as long as the first character is a letter. Spaces are not allowed, so use an underscore (_) to separate words. Actions are case-sensitive, so `STATUS_LIGHT_ON` is different than `status_light_on`. When naming Actions, choose a name that is self-explanatory.

NOTE! ***C/C++ keywords (such as if, else, or while) cannot be used as a name.***

Actions can be stored (S) or Unstored (N) after execution. An N or S appearing in the first cell of the action object indicates this attribute. Stored Actions keep the values of variables for the next time the step is active. Unstored Actions reinitialize variable values each time the step with which they are associated becomes active.

To add Actions:

1. Select the **[Action]** tool.
2. Click on the Step to which you want to add Actions. The **Add Action** dialog box appears.
3. Enter the name of the Action you want you want to add to the Step.
4. Select **Stored** or **Unstored**.
5. Click **[OK]**. The Action name now appears next to the Step with which it is associated in the Program.

To add content to the Action, double-click on the name of the Action. A Program window appears. This allows you to program the Action in any of the three programming languages.

To change the **Name** or **Stored/Unstored** attribute:

1. Select the **[Selector]** tool.
2. Double-click on the **Action**. This opens the Action's program window.
3. Select **Edit**.
4. Select **Attributes**.
5. Selection **Action...**
6. Edit the **Name** field, if desired.
7. Select the desired **Stored/Unstored** option.
8. Click **[OK]** to save the change.
   – or –
   Click **[Cancel]** to abort changes.

Comments

The **[Comments]** tool allows you to document each Step, to note variables, inputs, outputs, and other information pertaining to the Step. You can then view and edit this text as needed.

To enter comments:

1. Select the **[Comments]** tool.
2. Double-click on the Step you want to document. The **Comments** dialog box appears.
3. Enter text.
4. Click **[OK]** to save the text and close the dialog box.
   – or –
   Click **[Cancel]** to delete the comment.

Cross Reference

Cross Referencing provides both on-line viewing and reports of the variables defined as part of the SFC program. This helps you document and debug your control program.

NOTE! ***For more information on Cross Referencing, consult Chapter 16, Cross Referencing.***

8

Ladder Diagrams

The Ladder diagram is an outgrowth of the ‘relay Ladder’ commonly used in the programming of PLCs. This offers you a visual method of creating Boolean expressions. By satisfying the elements of the expression power flows through to a *coil* at the output of a rung.

Simply put, the Ladder diagram symbolizes a circuit with energized left and right power rails. If conditions exist that allow contacts, positioned along a rung of the Ladder, to close then the circuit is complete and power flows to an output coil. By placing several contacts in series you can create an AND condition in Boolean Logic. Conversely, an OR condition results from placing contacts in parallel with each other. The Ladder logic executes from top to bottom and from left to right. Although *LogicPro* does not draw the right power rail on the screen, the circuit functions as if it were there.

This programming technique provides you with a variety of contacts and coils, thus giving you considerable flexibility in creating intricate program logic. This chapter explains the various contacts and coils accessible to you in *LogicPro* along with some insight on how to use them. If you have ever programmed a ‘relay Ladder’ you are already familiar with most of these concepts and symbols.

To select Ladder diagram as the language for a program window, select **LD - Ladder** from the **Tools, Language** menu. However, if the program window has existing Sequential Function Chart or Function Block Diagram elements, a language change will not be allowed.

Ladder Toolbar

When you select Ladder diagram as the language for a program window, *LogicPro*:

- Displays the Ladder toolbar on the right side of the screen.
- Changes the menu items in the **Tools** menu.

To use any one of these tools, click it on the toolbar or select it from the **Tools** menu.

Table 8.1 contains a list of the available tools.

Table 8.1 Ladder Diagrams Toolbar

This Tool	Performs This Function
	Selector
	Normally Open Contact
	Normally Closed Contact
	Off to On Transitional Contact (Positive)
	On to Off Transitional Contact (Negative)
	Normal Coil
	Latched Coil (Set)
	Unlatched Coil (Reset)
	Function Block
	Comments
	Cross Reference

Selector

Use the **[Selector]** tool to select an element within a Ladder program. Once selected you can move, edit, delete or assign Variable names to it.

To use the Selector:

1. Select the **[Selector]** tool.

Select any element in the program by clicking once on it. This will cause handles to appear showing that it is selected. Clicking twice on it causes the **Variables** dialog box to appear.

Figure 8.1 illustrates an element without handles.

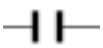


Figure 8.1 Element without Handles

Figure 8.2 shows a selected element with handles.

Figure 8.2 Selected Element with Handles

Normally Open Contact

The **[No Contact]** tool enables you to insert normally open contacts into the program window.

During a single evaluation of the Ladder diagram, if the Variable associated with this contact is True, then the contact will close. Thus allowing power to pass to the next element on the rung. If the associated Variable is False, the contact remains in the Open state.

To use the Normally Open Contact:

1. Select the **[Normally Open Contact]** tool.
2. Position the pointer where you want the contact placed and click.

Normally Closed Contact

The **[Normally Closed Contact]** tool allows you to insert normally closed contacts into the program window.

During a single evaluation of the Ladder diagram, if the Variable associated with this contact is True, then the contact will open. Thus preventing power to pass to the next element on the rung. If the associated Variable is False, the contact remains in the Closed state, which means power will pass when the element is False or (0).

To use the Normally Closed Contact:

1. Select the **[Normally Closed Contact]** tool.
2. Position the pointer where you want the contact placed and click.

Off To On Transitional Contact

The **[Off To On Transitional Contact]** tool allows you to insert off to on transitional contacts into the program window.

The Off to On Transitional Contact is sometimes known as the Positive Transitional Contact.

During a single evaluation of the Ladder diagram, if the Variable associated with this contact changes from False to True, then the contact will close for this cycle only. Thus allowing electricity to flow to the next element on the rung. For all other conditions this contact remains in the Open state.

To use the Off To On Transitional Contact:

1. Select the **[Off To On Transitional Contact]** tool.
2. Position the pointer where you want the contact placed and click.

On To Off Transitional Contact

The **[On to Off Transitional Contact]** tool allows you to insert on to off transitional contacts into the program window.

The On To Off Transitional Contact is sometimes known as the Negative Transitional Contact.

During a single evaluation of the Ladder diagram, if the Variable associated with this contact changes from True to False, then the contact will close for this cycle only. Thus allowing electricity to flow to the next element on the rung. For all other conditions this contact remains in the Open state.

To use On to Off Transitional Contact:

1. Select the **[On to Off Transitional Contact]** tool.
2. Position the pointer where you want the contact placed and click.

Normal Coil

The **[Normal Coil]** tool allows you to insert normal output coils into the program window.

The Normal Coil is set to On when all of the contacts on the Ladder rung pass power to the coil from the left power rail. This coil or Function Block remains On until one of the contacts opens and stops sending power to the coil.

To use the Normal Coil:

1. Select the **[Normal Coil]** tool.
2. Position the pointer where you want the coil placed and click.

Latched Coil

The **[Latched Coil]** tool allows you to insert latched coils into the program window

The Latched Coil, sometimes known as a Set coil, is set to On when all of the contacts or Function Blocks on the Ladder rung pass power to the coil from the left power rail. This coil remains *latched* On until it is reset by a Reset (Unlatched) coil.

To use the **[Latched Coil]**:

1. Select the **[Latched Coil]** tool.
2. Position the pointer where you want the coil placed and click.

Unlatched Coil

The **[Unlatched Coil]** tool allows you to insert unlatched coils into the program window.

The Unlatched Coil, sometimes known as a Reset coil, is reset to Off when all of the contacts or Function Blocks on the Ladder rung pass power to the coil from the left power rail. The Unlatched Coil is given the same Variable name as the corresponding Latched Coil. When the Unlatched Coil is energized, the corresponding Latched Coil is de-energized or *unlatched*.

To use the Unlatched Coil:

1. Select the **[Unlatched Coil]** tool.
2. Position the pointer where you want the coil placed and click.

Function Block

The **[Function Block]** tool allows you to insert Function Blocks into the program window.

Function Blocks are useful in Ladder diagram programs to perform integer or real mathematics, counter or timer functions, or other higher functions.

Function Blocks must have unique names and their I/O must be assigned. I/O may be constants, variables or rung connections.

Refer to *"Inserting Function Blocks" on page 89* for detailed instructions on adding Function Blocks to Ladder diagrams.

Comments

The **[Comments]** tool allows you to type comments into the **Rung Comments** dialog box. With this feature, you can document each rung in the Ladder diagram to:

- Note each element present.
- Define its purpose in the control program.
- You can view and edit this text as needed.

To use Comments:

1. Select the **[Comments]** tool.
2. Position the pointer within the box formed by the grid points that includes the number of the rung you want to comment, and click. This opens the **Comments** dialog box.
3. Type your comments.
4. Click **[OK]** to save your comments.
   – or –
   Click **[Cancel]** to close the dialog box without saving your comments.

Cross Reference

The Cross Reference tool allows you to obtain a list of the other occurrences of program elements you click on.

The *LogicPro* Cross Referencing feature provides on-line viewing and report print-outs of the Variables defined as part of the Control program. This helps you document and debug your process program.

NOTE! ***For more information on Cross Reference feature, consult Chapter 16, Cross Referencing.***

Viewing the Grid

You can enable the Grid menu option to display a grid of dots in the program window. This grid is helpful when connecting or operating on program elements.

To use the Grid feature:

1. Select **View**.
2. Select **Grid**. This enables the Grid command.

Creating and Editing Ladder Diagrams

LogicPro allows you to use the many editing features available in Windows.

Inserting Rungs

To insert a rung into a Ladder program between two existing rungs:

1. From the toolbar select the new element type you want to add.
2. Click: Anywhere in the program window to create the first rung
   - or -
   Between two existing rungs
   - or -
   Above the first rung
   - or -
   Below the last rung.

The program is redrawn with the element on a new rung. The rung number in front of the left power rail is updated.

NOTE! ***In rungs with multiple branches, such as the one in Figure 8.3, a new rung cannot be inserted between these branches.***

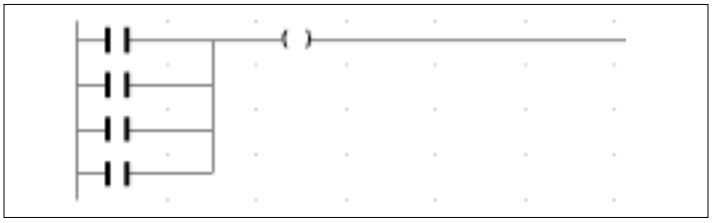


Figure 8.3 Rungs with Multiple Branches

Inserting Elements

To insert an element on an existing rung:

1. Select the element type you want to add from the toolbar.
2. Click on the rung.

The new element is inserted and the adjacent elements adjust themselves to accommodate the new element.

NOTE! ***The coil must always be the last element on a rung. The editor issues a warning when a user attempts to put the coil anywhere else.***

Inserting Function Blocks

To add a Function Block to a Ladder diagram:

1. Select the **[Function Block]** tool. The **Function Block** dialog box appears, as shown in *Figure 8.4*.
2. Select the type of Function Block you want:
   Standard
   - or -
   Vendor Provided
   - or -
   User Defined
3. Select the name of the Function Block you want from the list.
4. Click **[OK]**.
5. Click the location on the screen where you want to insert the Function Block. The Function Block appears in that location. See *Figure 8.5* and *Figure 8.6 on page 90.*

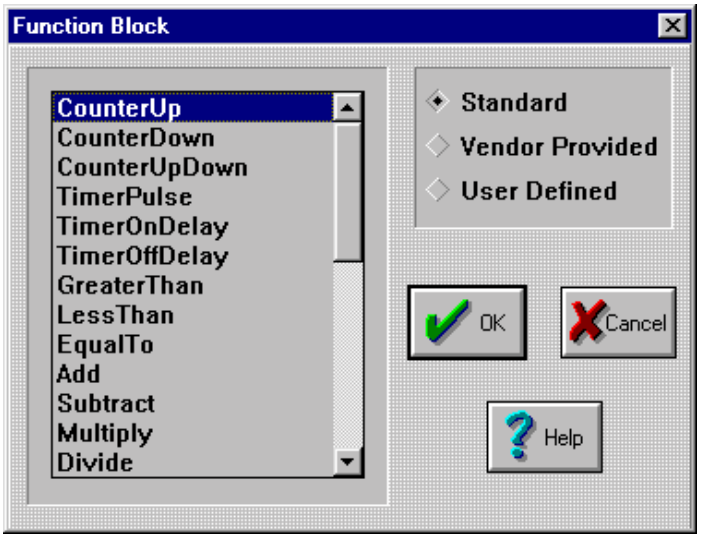


Figure 8.4 Function Block Dialog Box

You can place Function Blocks on an existing rung with elements on either side, as shown in *Figure 8.6 on page 90*, or you can insert the block along with a new rung. In either case,

the surrounding rungs and elements shift to accommodate the newly added Function Block.

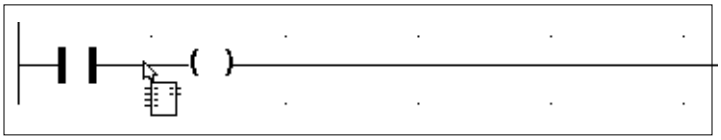


Figure 8.5 Rung Before Adding a Function Block

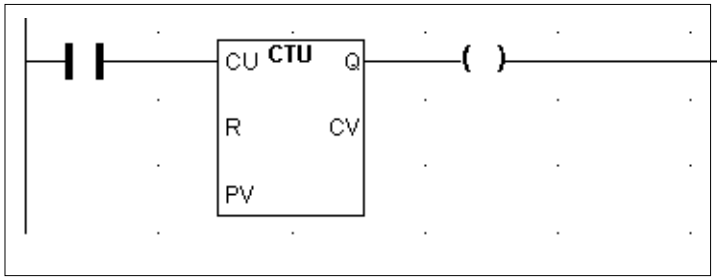


Figure 8.6 Rung With a Function Block Added

NOTE! ***More detailed information on Function Block Diagrams and Function Blocks is available in Chapter 9 and Chapter 10, respectively.***

Connecting Function Blocks to Other Ladder Elements

You can add Ladder elements, such as contacts and coils, on both sides of a Function Block. *Figure 8.7 on page 91* shows a Ladder diagram before additional elements are connected to the Function Block, while *Figure 8.8 on page 91* shows the same Ladder diagram with the elements now included.

To connect Function Blocks to other Ladder elements:

1. Select the element type you want to add from the toolbar.
2. Position the pointer on the Ladder diagram where you want to add the element, then click.

NOTE! ***Coils cannot be used on the input side of a Function Block.***

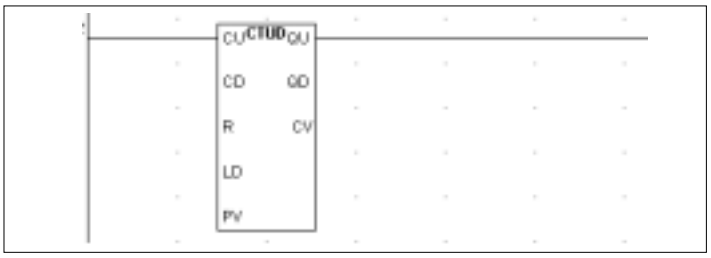


Figure 8.7 Ladder Diagram Before Elements Are Added to Function Block

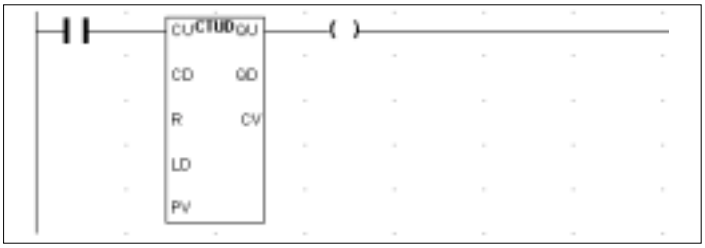


Figure 8.8 Ladder Diagram After Elements Are Added to Function Block

Function Blocks with More Than One Input and Output Bit

Some Function Blocks have more than one input and output logical bit. For example, the CTUD (Counter Up Down) Function Block has four logic input bits (CU, CD, R, LD) and two logical bits (QU and QD) on the output side, as shown in *Figure 8.9 on page 92*.

To insert a new element between the left power rail and the second or lower logical input:

1. Select the element type from the toolbar.
2. Click and hold on the right side of the left power rail directly across from the logical input of the Function Block. Handles appear on the left power rail to indicate that you have selected the correct part of the rail.
3. Drag the line to the connecting point on the Function Block.
4. Release the mouse button. A rung appears with the element on it, as shown in *Figure 8.10 on page 92*.

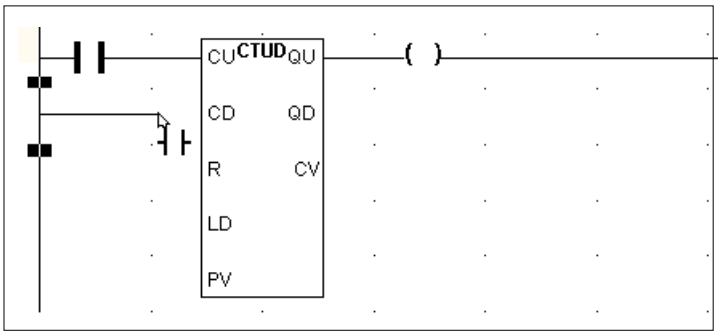


Figure 8.9 Inserting an Input Element on a Function Block

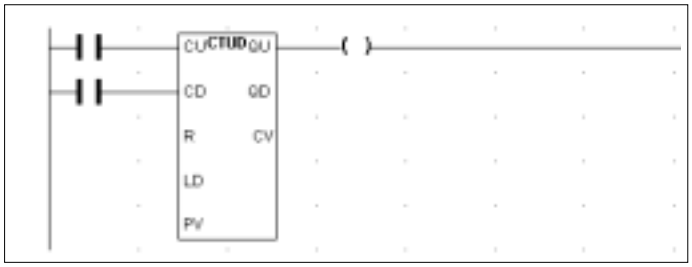


Figure 8.10 Element Inserted Between Left Power Rail and Second Input Bit

Connecting New Elements

To connect a new element, such as a coil, from the second or lower logical output bit to the right power rail:

1. Select a coil element type from the toolbar.
2. Click and hold on the Logical Output Bit in the Function Block.
3. Drag the line at least past the grid that contains the right side of the Function Block. See *Figure 8.11 on page 93*.
4. Release the mouse button. A rung appears with the new element on it. See *Figure 8.12 on page 93*.

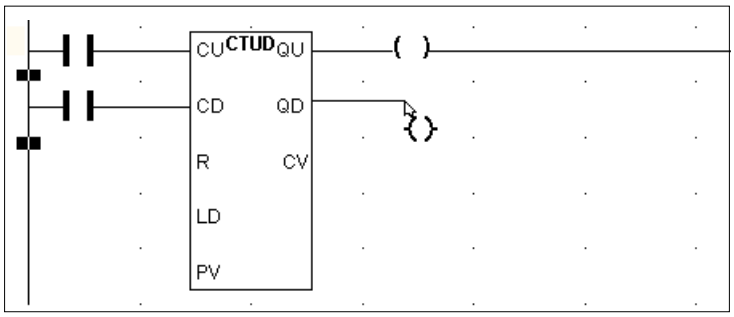


Figure 8.11 Connecting an Element to a Function Block's Output

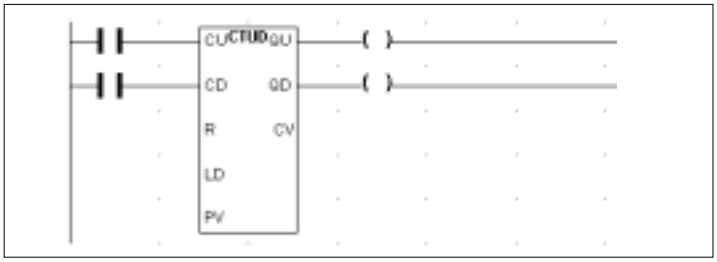


Figure 8.12 Function Block After Connecting Output Element

NOTE! ***If the Function Block's first logical input is connected directly from the left power rail without any element in between, a constant True condition is applied during the running of the control program. However, for the other logical input, if there is no connection between it and the left power rail, a constant False condition is applied.***

Inserting Parallel (Or) Branches

There are two ways you can add a Ladder element parallel to an existing element.

To create parallel branches, there must be elements on both sides of the element you want to parallel. *Figure 8.13 on page 94* shows the Ladder diagram before the parallel branch is added, while *Figure 8.14 on page 94* shows the Ladder diagram after the branch is added.

1. From the toolbar, select the element type you want to add.

2. Click and drag from the left side to the right side of the element you want to parallel.
3. Release the mouse button. A parallel branch is created.

NOTE! ***When clicking and dragging you must begin or end outside the grid cell that contains the element you want to parallel.***

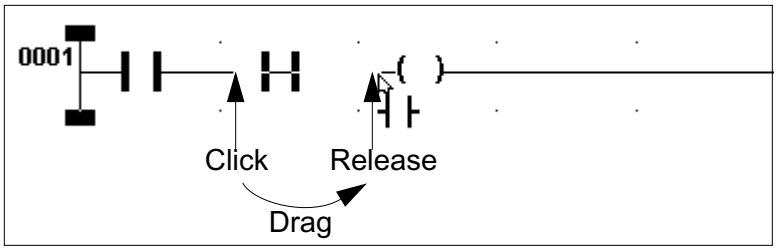


Figure 8.13 Adding the Parallel Branch

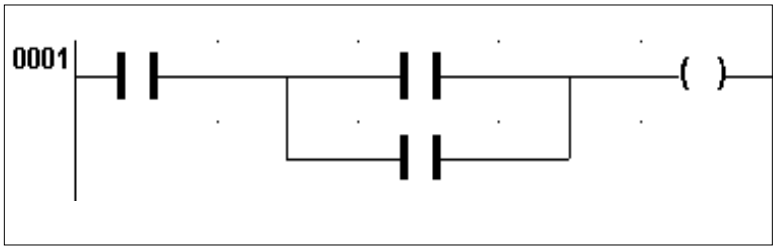


Figure 8.14 After Adding the Parallel Branch

If more parallel branches are needed, as shown in *Figure 8.15 on page 95*:

1. From the tool bar, select the element type to add.
2. Click and drag from the left side to the right side of the element you want to parallel.
3. Release the mouse button with the pointer in the grid cell to the right of the cell in which the element to parallel is located.

The finished diagram is shown in *Figure 8.16 on page 95*.

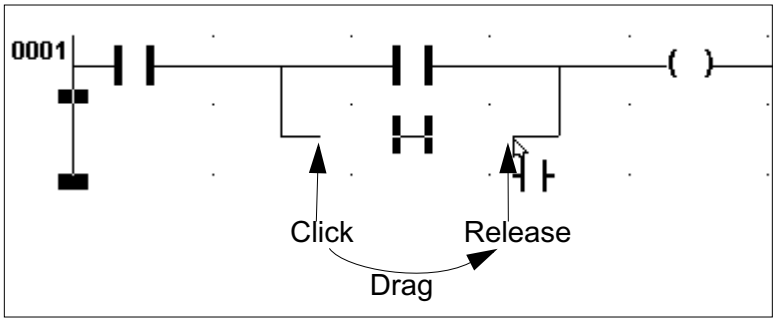


Figure 8.15 Adding Another Branch

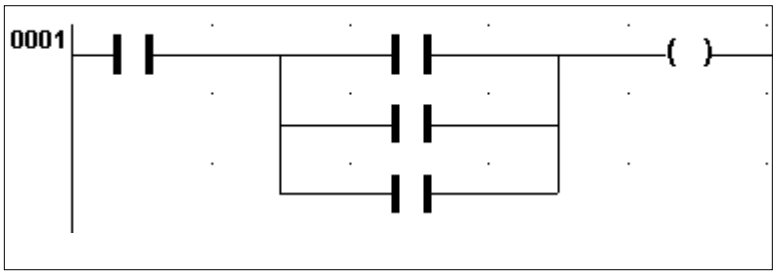


Figure 8.16 Branch Added

To create a new parallel branch from the left power rail, as shown in *Figure 8.17 on page 96*:

1. From the toolbar, select the element type you want to add.
2. Start dragging on the right side of the left power rail and below the rung you to which you want to attach the new branch.
3. End dragging close to the rung.

The finished diagram is shown in *Figure 8.18 on page 96*.

NOTE! ***Ensure that the handle does not appear on the rail below, if there is one. This is to ensure that you want to insert a new rung.***

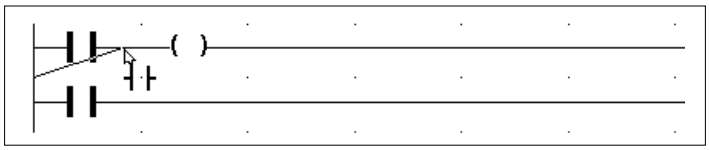


Figure 8.17 Creating a New Parallel Branch from the Left Power Rail

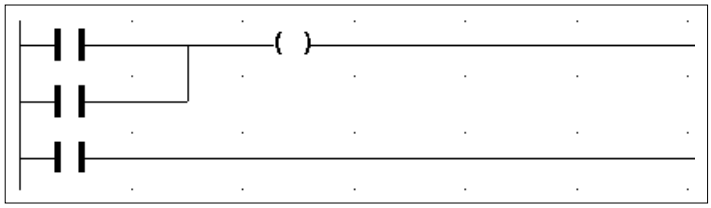


Figure 8.18 After Creating a New Parallel Branch from the Left Power Rail

If you need to add more parallel branches:

1. From the tool bar, select the element type to add.
2. Start dragging on the right side of the left power rail and below the rung you to which you want to attach the new branch.
3. End dragging close to the rung.

NOTE! ***Ensure that the Handles appear on the rail directly below the rung number to ensure that the new parallel OR branch starts from the existing rail.***

Inserting Multiple Output Coil branches

To create multiple output coil branches:

1. Select a coil type element (either normal coil, latched coil or unlatched coil) from the toolbar.
2. Click and drag from the left of the coil below and past the existing coil. See *Figure 8.19 on page 97*.
3. Release the mouse button. A new branch with the new coil is added below the original. See *Figure 8.20 on page 97*.

NOTE! ***This procedure is valid only when the selected element type is one of the three output coil types. However, other element types may be placed before the coil in a multiple output coil branches.***

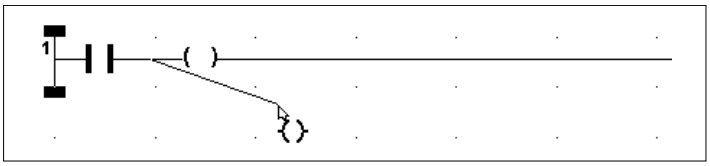


Figure 8.19 Before Creating Multiple Output Coil Branches

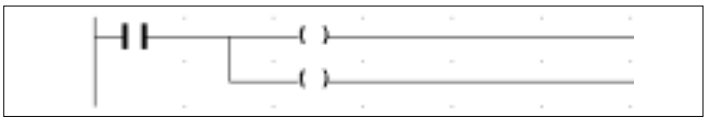


Figure 8.20 After Creating Multiple Output Coil Branches

NOTE! ***If the top most coil in a multiple output coil branches is deleted, a horizontal line will remain, as shown in Figure 8.21. This should not be a cause of concern as it is only the graphical representation and does not mean a 'short circuit' in the actual control logic.***

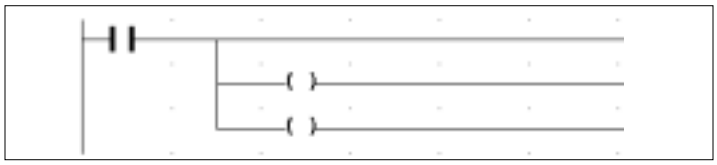


Figure 8.21 Multiple Output Coil Branches after the Top Coil is Removed

Inserting Elements From the Left Power Rail

To create an extension from the left power rail to existing parallel OR branches or multiple output coil branches:

1. Make sure that the extension you want to add will not get in the way of any existing element. *Figure 8.22* shows a Ladder diagram that is ready to accept an extension rail.
2. Select the new element type from the toolbar.
3. Click and hold on the right side of the left power rail. Handles appear to indicate that you selected an existing rail.
4. Drag just to the right of left vertical line of the branch to which you want to attach.
5. Release the mouse button. A new extension rail is created, as

shown in *Figure 8.23*, along with the new element.

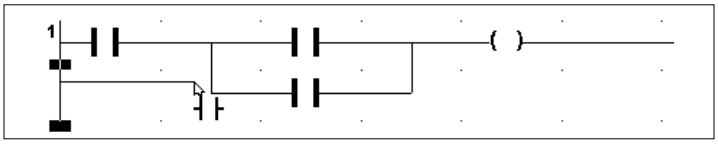


Figure 8.22 Adding an Extension from the Left Power Rail

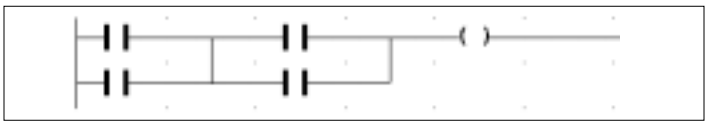


Figure 8.23 After Adding an Extension from the Left Power Rail

Cutting Rungs

To cut a rung:

1. Select the **[Selector]** tool.
2. Position the pointer to the left of the Left Power Rail (each Rung has its own unique number)
3. Click on the number of the Rung number that you want to Cut. You may select multiple Rungs by holding down the left mouse button and dragging the "dotted box" around adjacent Rung numbers.

4. Click **[Cut]** on the Standard toolbar.
   – or –
   Select **Edit,** then **Cut**
   – or –
   Press **<Ctrl> + <X>.**

The Selected Rung(s) disappear(s) from the screen. See *"Pasting Rungs"* below for directions on pasting the elements elsewhere.

Copying Rungs

To copy a rung:

1. Select the **[Selector]** tool.
2. Position the pointer to the left of the Left Power Rail (each Rung has its own unique number).
3. Click on the number of the Rung number that you want to Copy. You may select multiple Rungs by holding down the left mouse button and dragging the "dotted box" around adjacent Rungs numbers.
4. Select **[Copy]** on the Standard toolbar.
   – or –
   Select **Edit**, then **Copy**
   – or –
   Press **<Ctrl> + <C>.**

The Selected Rung(s) are copied and available for pasting, see *"Pasting Rungs"* for directions on pasting the elements elsewhere.

Pasting Rungs

To paste a rung:

1. Click **[Paste]** on the Standard toolbar.
   – or –
   Select **Edit**, then **Paste**
   – or –
   Press **<Ctrl> + <P>.**
2. The Selector pointer changes to indicate the next click will paste a rung.
3. Position the pointer where you want to Paste the Rung.
4. Click. *LogicPro* inserts the Rung(s) at this point. You can continue to Paste the same Rung(s) until you overwrite this data with another Cut or Copy instruction.

Deleting Elements

Individual or multiple elements may be deleted. Deleting all the elements on a rung deletes the rung.

To Delete an element or group of elements from the programming window:

1. Select the **[Selector]** tool.
2. Click on the element that you want to Delete. You may select multiple elements by holding down the left mouse button and dragging the "dotted box" around adjacent elements.
3. Select **[Delete]** from the Standard toolbar
   -or-
   Press the **Delete** key.

Moving Elements

Select the element. Click and hold the left mouse button and drag the element to its destination.

Rungs may be Moved by Cutting and Pasting. Elements cannot be Moved from one rung to another. Instead they must be Deleted and recreated where desired.

Assigning Labels to Rungs

Rungs can be labeled to facilitate logic jumps Rung labels may consist of up to six mixed case alphanumeric characters and the underscore character. Labels are visible whether comments are displayed or not.

To Label a rung:

1. Double-click the rung number with the **[Selector]** tool. The **Set Rung Label** dialog box appears.
2. Type a unique Label in the **Label for Rung #** field.
3. Click **[OK]** to save the label
   – or –
   [Cancel] to abort labeling.

Assigning Variable Names

To assign a Variable name to an element:

1. Click the **[Selector]** tool.
2. Double click on the element to which you want to assign a Variable name. The **Variables** dialog box appears, as shown in *Figure 8.24*.
3. You can view System, Project, Resource, and Program Variables by checking the appropriate box.
4. Select the Variable from this list or create a new Variable using the **[Add]** or **[Clone]** buttons (see *Chapter 6, "Variables"*).
5. Click **[OK]**.

The Variable name appears over the Ladder element in the program. You can change the Variable assignment by double-clicking on the element and selecting another Variable from the list.

NOTE! ***All Ladder elements must be assigned Variable names. If an element is not assigned a Variable name, an error occurs during the program code generation process.***

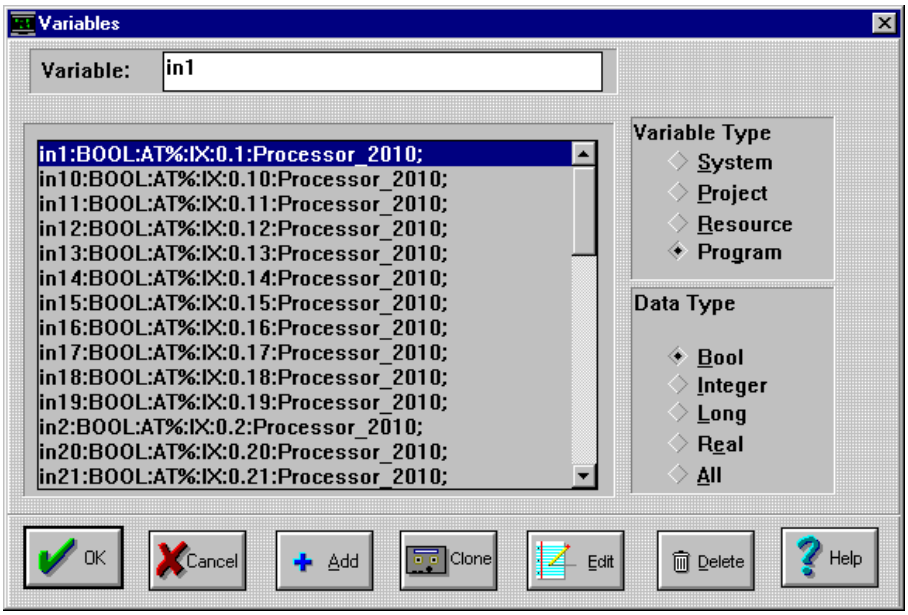


Figure 8.24 Variables Dialog Box

Naming and Assigning Variables to Function Blocks

Function Blocks must have unique names and their I/O must be assigned. I/O may be constants, variables or rung connections. Inputs and outputs may be integer or real or Boolean.

To assign symbolic Variable names or preset values to a function block:

1. Select the **[Selector]** tool.
2. Double-click on an existing function block. The **Assign Function Block Symbolic Input/Output** dialog box for that function block appears, as shown in *Figure 8.25 on page 103*. The Input and Output to which you can assign Variables are listed in the dialog box.
3. Enter a unique name in the **FB Name** field. If necessary, click **[List]** to see a list of Function Block names in the current program.
4. Select the value in the **Symbolic Input** or **Symbolic Output** field to which you want to assign a value.
5. Enter a numeric value or variable name in the field at the bottom of the dialog box,
   – or –
   click the **[Search]** button and double-click on a variable name in the displayed list.
6. Repeat steps 4 and 5 until all Symbolic I/O are assigned.

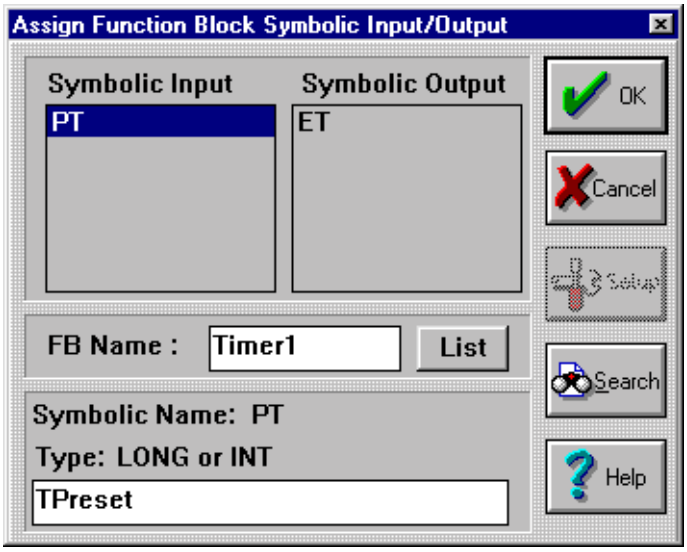


Figure 8.25 Assign Function Block Symbolic Input/Output Dialog Box

7. Click **[OK]**. The symbolic names or values appear on the function block in a Ladder diagram program, as shown in *Figure 8.26*.

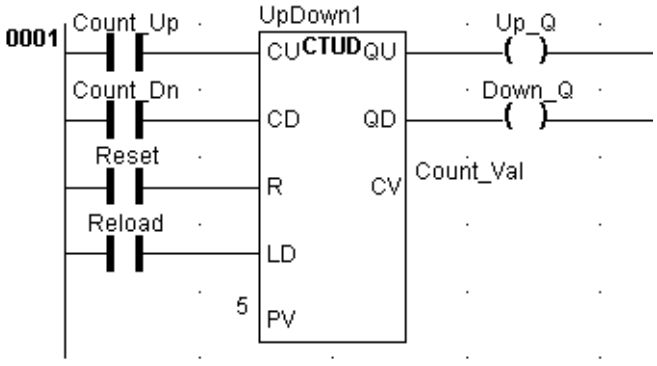


Figure 8.26 Symbolic Name on Function Block in a Ladder Diagram Program

NOTE! ***The outputs of a Function Block need not be attached to Ladder elements.***

Adding Comments to Ladder Diagrams

To add comments to a rung in the Ladder diagram:

1. Select the **[Comment]** tool.
2. Click on the number of the rung to which you want to add comments. The **Rung Comment** dialog box appears, see *Figure 8.27*.
3. Type in the comment.
4. Click **[OK]** to save the text and close the dialog box
   – or –
   Click **[Cancel]** to terminate the procedure.

The comment is displayed immediately above the rung you documented.

NOTE! ***Any blank line left at the end is truncated in order to conserve space.***

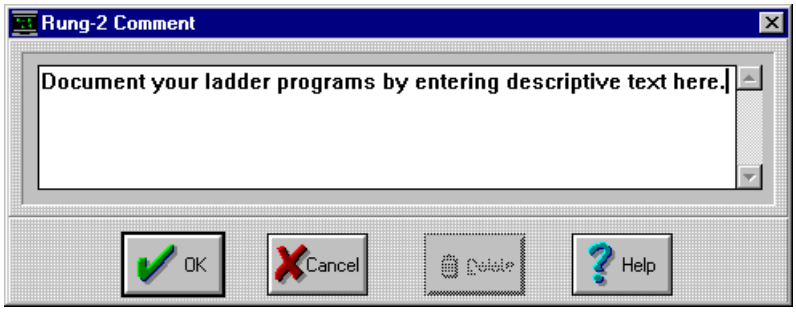


Figure 8.27 Rung Comment Dialog Box

Hiding Rung Comments

You may occasionally want to have a more compact Ladder diagram so that you can view more Ladder logic within a program window. In this case, to hide all Rung Comments:

1. Select the **View** menu.
2. Deselecting **Comment** command
   – or –
   select the **[Comment]** tool.

9

Function Block Diagrams

Function Block diagrams (FBD) is a graphical programming language provided by *LogicPro*. In the broadest sense, this language allows you to manipulate function blocks in much the same way as in the Ladder programming language. As with the Ladder language, FBD allows you to simply create and edit complex programs using a powerful set of tools.

The biggest difference between the Ladder and FBD languages is their focus. While Ladder allows you to create complex programs that include function blocks, FBD is simpler to use for applications such as creating process programs where inputs tend to be something other than discrete devices such as limit switches and proximity sensors. FBD allows you to map out what comes in and where it's coming from without being restricted by the rigid structure of a discrete oriented Ladder diagram.

As with a Ladder program, an FBD program can be used either inside an action of an SFC program or as a stand-alone program.

FBD Toolbar

Table 9.1 shows the SFC tools with a brief description of each tool's function. The following sections describe the tools and the Actions they produce in greater detail.

Table 9.1 FBD Toolbar Functions

Tool	Description
	Selector
	Straight Connector
	Inverted Connector
	Variables
	Function Block
	Cross Reference

Selector

The **[Selector]** tool allows you to choose one or more element of your program for editing. Clicking on this button makes the pointer appear as the traditional arrow. Using the pointer, you can either click on a single element or you drag around a group of elements you wish to modify.

This is the default tool for FBD. When you create a new program using the FBD language, this tool is already selected.

Straight Connections

The **[Straight Connections]** tool is used to make input and output connections between Function Blocks and Variables. Use this tool to drag from any element or portion of an element to any another. Selecting this tool makes your pointer look like the traditional arrow with a line-arrow attached to the bottom of it.

Inverted Connections

The **[Inverted Connections]** tool is almost identical to the Straight Connections tool. The differences are:

Both elements to be connected using this tool must represent Boolean (logical) data.

Whatever the state at the source of the connection the data will be inverted at its destination (if it was on/true at the source, it will be off/false at the destination)

Selecting this tool makes your pointer look like it does for Straight Connections, except the arrow attached to the bottom of your pointer has a circle at its point.

Variables

The **[Variables]** tool allows you to insert new Variable elements into your program. Selecting this tool makes your pointer take on the appearance of a traditional arrow with the addition of a small symbol identical to the picture on the button. Clicking anywhere in the program window causes a new Variable element to be added.

Function Blocks

The **[Function Block]** tool allows you to insert function blocks into the program window.

Cross Reference

Cross Referencing provides both on-line viewing and reports of the variables defined as part of the FBD program. This helps you document and debug your control program.

Creating an FBD Program

There are six basic steps to creating a program using FBD:

- Inserting Function Blocks
- Inserting Variable Elements
- Connecting Inputs and Outputs
- Assigning *LogicPro* defined Variables to Variable Elements
- Assigning Names to Function Blocks
- Saving, Compiling, Downloading, and Monitoring.

NOTE! ***Variable Elements must be connected before values can be assigned to them.***

Each of these steps are discussed in more detail in the following sections.

Inserting Function Blocks

To add a function block:

1. Select the **[Function Block]** tool. The **Function Block** dialog box appears, as shown in *Figure 9.1*.
2. Select the check box next to the type of function block you want:
   - **Standard**
   - **Vendor Provided**
   - **User Defined**
3. Select the name of the function block you want from the list.
4. Click **[OK]**.
5. Click the location on the screen where you want to insert the function block. The function block appears in that location.

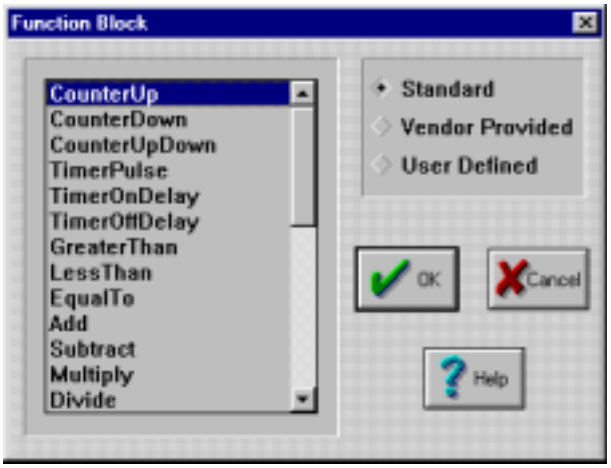


Figure 9.1 Function Block Dialog Box

Figure 9.2 shows two function blocks placed in the **Program** window.

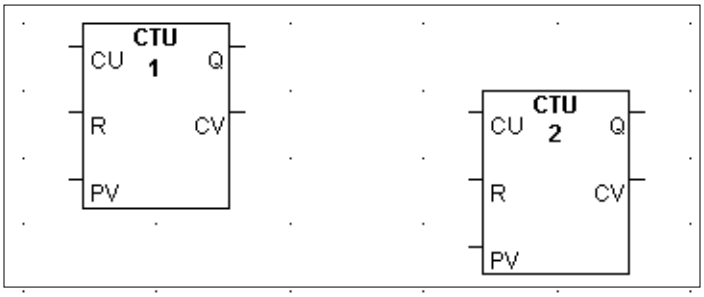


Figure 9.2 Placed Function Blocks

Inserting Variable Elements

To insert Variable elements:

1. Select the **[Variable]** tool.
2. Move the pointer to the location at which you want to place the new Variable element and click the mouse button.

NOTE! ***While function block outputs can be left open (without any Variables assigned to it), ALL INPUTS must have a Variable assigned to them.***

Connecting Inputs and Outputs

Connections are made using the Straight Connections and Inverted Connections tools. To make a new connection:

1. Select the type of connection you wish to make by clicking on the corresponding button.
   - Straight

     - or -
   - Inverted
2. Place your pointer over one of the elements you wish to connect, click and drag over to the element you wish to connect to and release the mouse button. See *Figure 9.3 on page 110.*

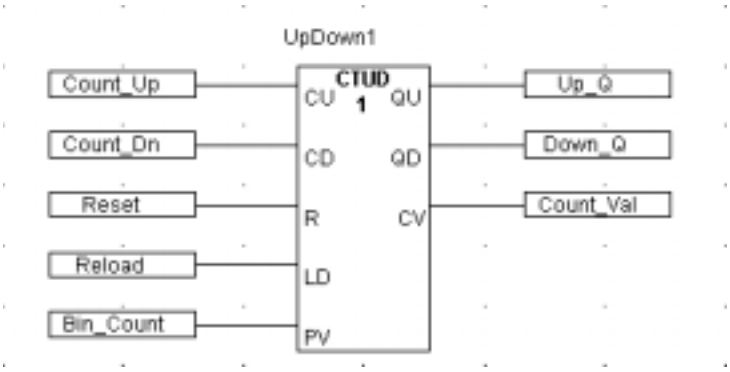


Figure 9.3 Symbolic Name on Function Block in a FBD program

NOTE! ***Branched connections are not currently allowed. Connections can only be made on a one-to-one basis; in other words, a Variable element can only be connected to one input or output.***

NOTE! ***Feedback loops to/from a single function block (connections which both originate and terminate on the same element or elements of a common function block) are not allowed. BUT, it is permissible to create a feedback loop to a system of function blocks.***

Assigning Variables to Variable Elements

To assign variables to Variable elements:

1. Select the **[Selector]** tool.
2. Double-click on the Variable element you wish to assign. This element must already be connected to either an input or output. This causes the **Variables** dialog box to appear.
3. From the dialog box, select the appropriate **Variable Type** and **Data Type**.
4. Either select the Variable you wish to assign to this Variable element or click on **[Add]** to create a new Variable.

For more information on creating new Variables, refer to *"Adding a Project Variable" on page 52,* or *"Adding a Resource/Program Variable" on page 56*

Assigning Names and Execution Orders to Function Blocks

Each function block in an *FBD* program must be assigned a unique name and execution order. To assign a name to a function block:

1. Select the **[Selector]** tool.
2. Double-click on the function block you wish to name. This causes the **Assign Name to Func. Block** dialog box to appear as shown in *Figure 9.4*
3. Enter a unique name for the function block into the **Name** field. You can click **[List]** to view all the function block names already assigned in the program.

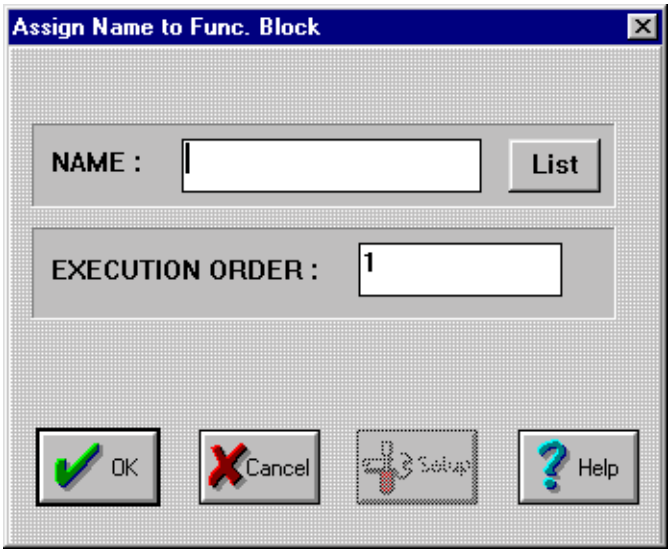


Figure 9.4 Assign Name to Function Block Dialog Box

NOTE! ***If you do not assign a name to a function block, LogicPro will assign a unique name to the new function block for you.***

NOTE! ***The maximum length of a function block name is ten (10) alphanumeric characters.***

4. Fill in the **Execution Order** field. An execution order is automatically assigned to the function blocks (determined by the order in which you created them) but it is also possible to change the execution order of the function blocks at any time.

Execution order is not bound by the apparent layout of the program. In the example shown in *Figure 9.5*, the function block named **two** is assigned to be first in the execution order, then **three**, and *then* **one**. While the CTU function block called **two** depends on **one** it can be executed first because all values are saved from the last cycle during which the program ran and those values are applied again. So, while it might seem strange that **two** runs first, being dependent upon input from **one**, the last value that was passed to **two** from **one** is used instead of new data.

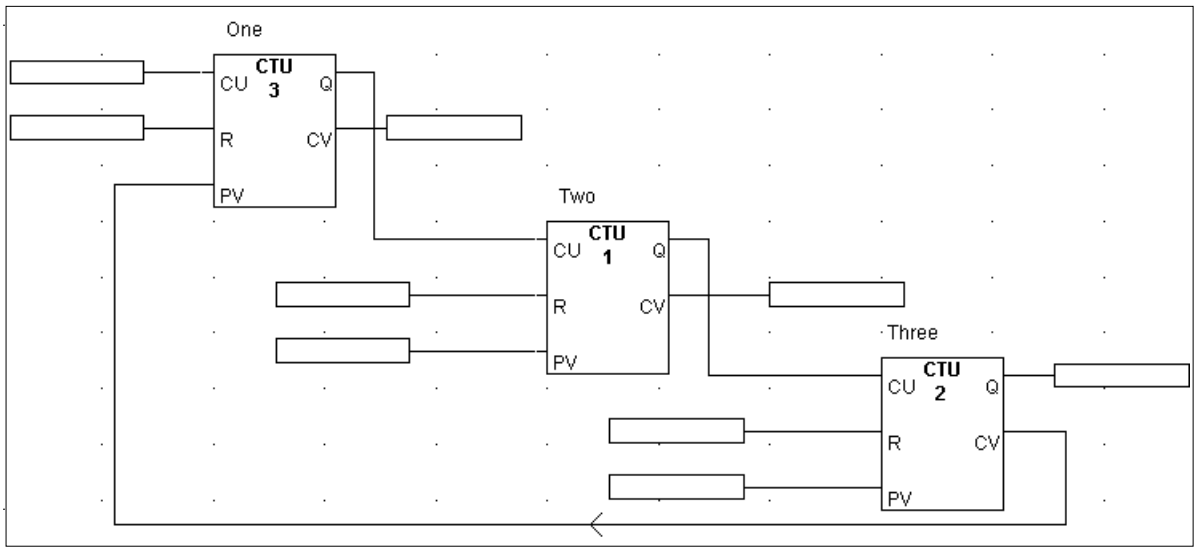


Figure 9.5 Execution Order in FBD

Cutting, Copying, Pasting, and Deleting

When cutting or copying a function block, it is necessary to make sure that all the function block's input and output Variables are also selected. You can tell that they are selected if they appear with "handles" around them, as shown in *Figure 9.6 on page 113*.

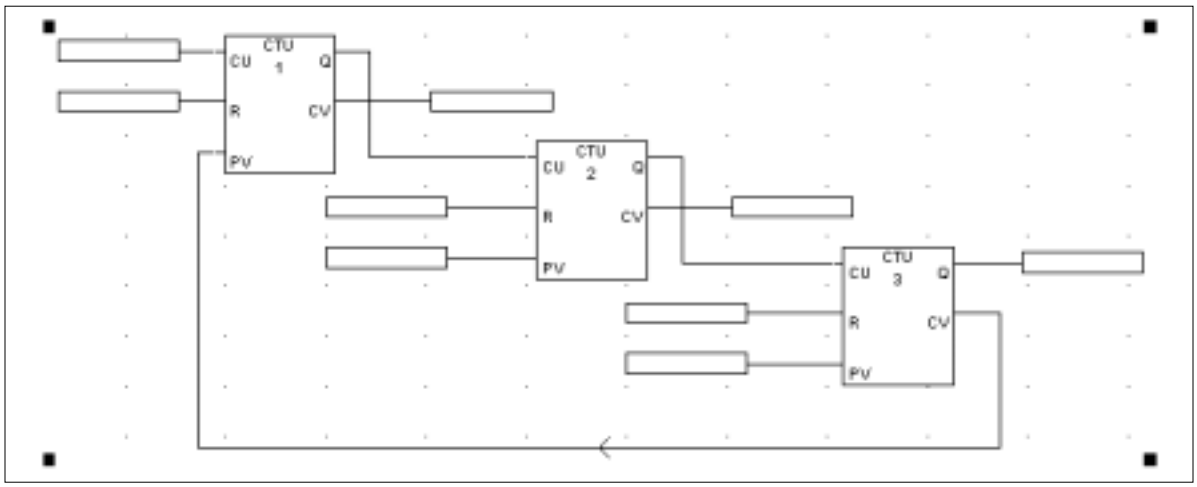


Figure 9.6 Function Blocks in FBD with Handles

NOTE! ***When cutting or copying, all inputs and outputs must be attached to something, be it a Variable or the input or output of another function block.***

When pasting a copy of a function block into the program from which it was copied, *LogicPro* automatically assigns it a new, unique name as shown in *Figure 9.7*.

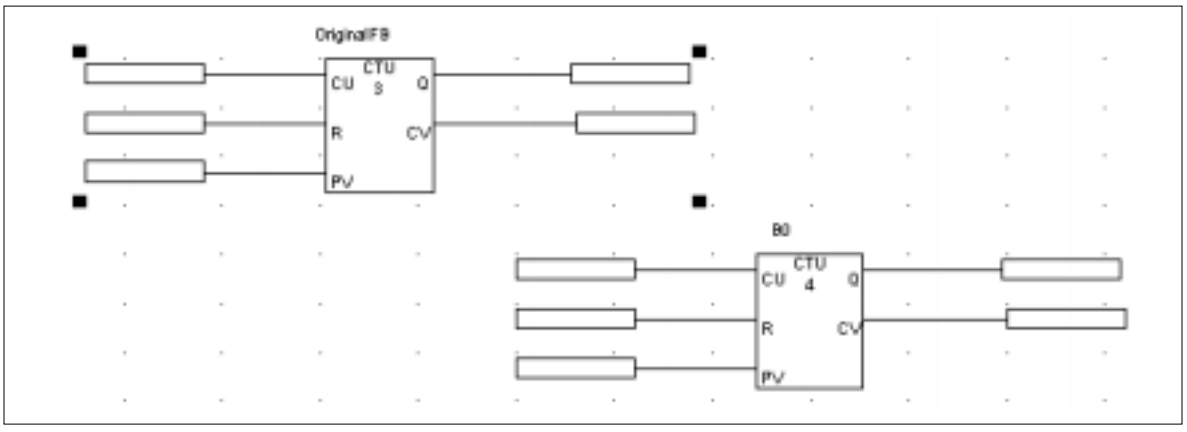


Figure 9.7 Pasted Function Block in FBD

When deleting a function block, all the Variables dependent upon that function block are deleted, as well as shown in *Figure 9.8* and *Figure 9.9 on page 114*.

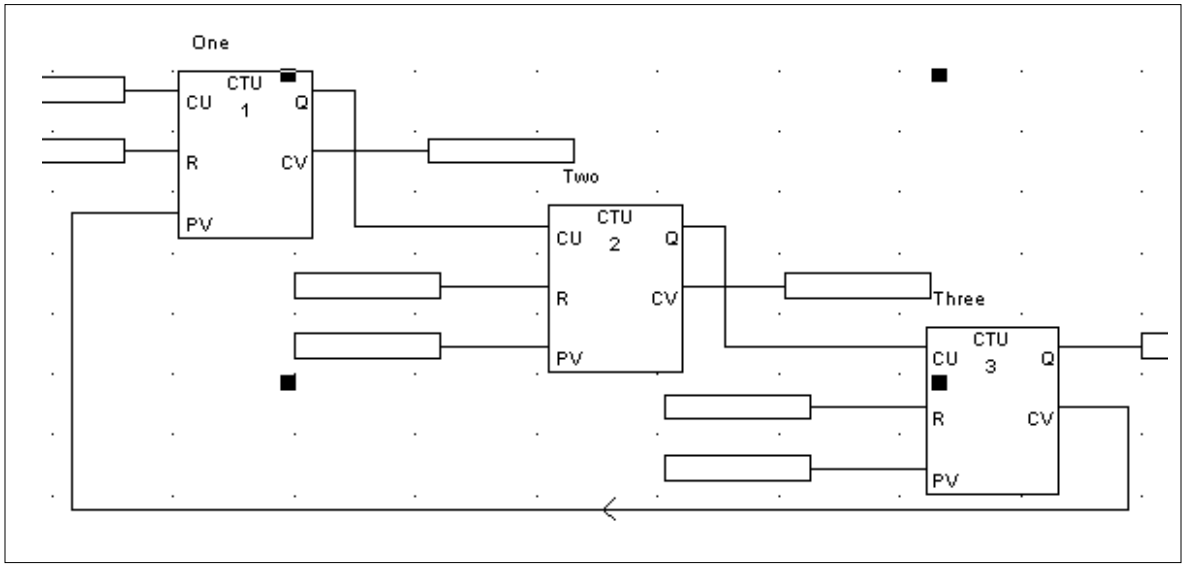


Figure 9.8 Function Block Before Deletion, With Handles

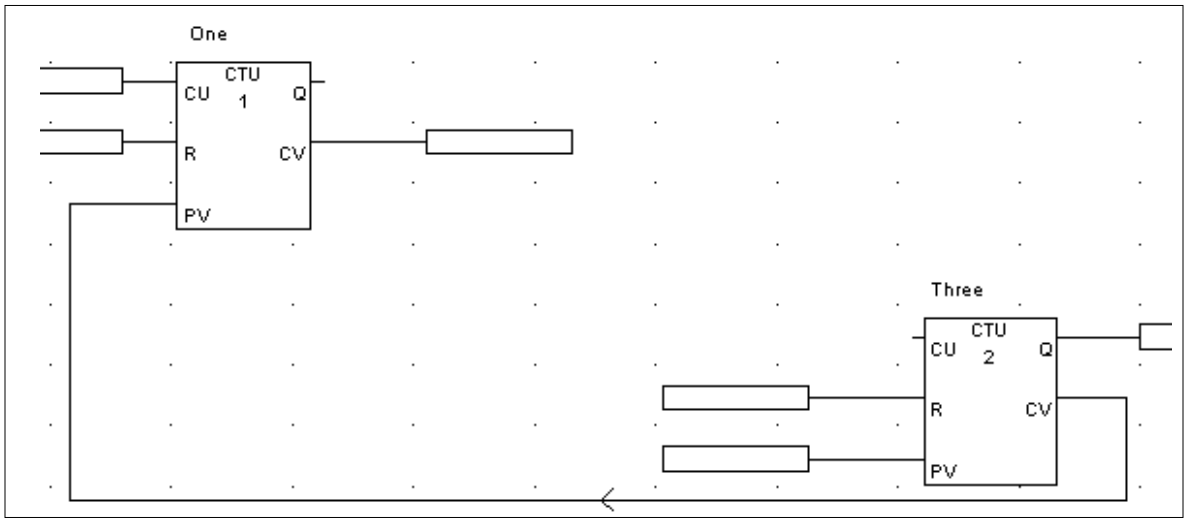


Figure 9.9 Function Block After Deletion

10

Function Blocks

This chapter explains in detail the function and I/O requirements for each of the standard and vendor provided Function Blocks used in FBD and LD programs. For information on writing your own Function Blocks, see *Chapter 11, "User Defined Function Blocks."*

The Function Block tool allows three types of function blocks to be inserted into Ladder Diagram and Function Block Diagram programs:

- Standard *LogicPro*
- Vendor Provided
- User Defined

Standard Function Blocks

The following sections describe the standard function blocks available in *LogicPro*. These standard function blocks fall into one of six categories:

- Timing and Counting
- Comparison
- Arithmetic
- Data Manipulation
- Real Number
- Logical

Timing and Counting Elements

The following sections contain descriptions of timing and counting function blocks found in the *standard* category.

Counter Up (CTU)

The Counter Up function block shown in *Figure 10.1*, enables an output after a preset number of transitions from False to True occur.

The CTU block increments the integer current value (CV) for every False to True transition of the counter up enable bit (CU). The current value continues to increment until it reaches the integer preset value (PV). At that point, the output bit (Q) is enabled and remains enabled until the function block is reset by a True condition of the reset bit (R). While the reset bit is True, the current value resets and remains at zero.

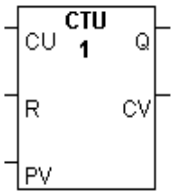


Figure 10.1 Counter Up Function Block

Inputs

CU is the Counter Up enable bit. As long as the reset bit is False, each False to True transition of this bit increments the current value until the current value equals the preset value. In a program using Ladder logic, the value of CU is determined by the Boolean evaluation of elements placed between the left power rail and CU in the function block. In FBD programs, the value of CU is determined by the Variable connected to CU. If no contacts are placed on the rung before CU, the value of CU is always True, but a transition never occurs.

R is the counter up reset bit. When this bit is True, the current value is set to zero and the output bit is disabled. The value of R is determined by the Boolean evaluation of elements placed between the left power rail and R in the function block. If no connection is made between the left power rail and R, the value of R is always False.

PV is the counter up preset value. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from 1 to 32,767 for **INT** type Variables and 1 to 2,147,483,647 for **LONG** type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

Q is the Counter Up output bit. This bit is enabled and passes power when the current value is equal to the preset value.

CV is the Counter Up current value. In Ladder logic, assign this output Variable a symbolic name in the **Assign Function Block Symbolic Input/Output** dialog box, shown in *Figure 10.2*. In FBD, assign a variable to this output in the **Variable Assignment** dialog box. The value of this Variable increments by one every time a False to True transition of the counter up enable bit occurs until the current value equals the preset value. The current value retains its value until the reset bit is True.

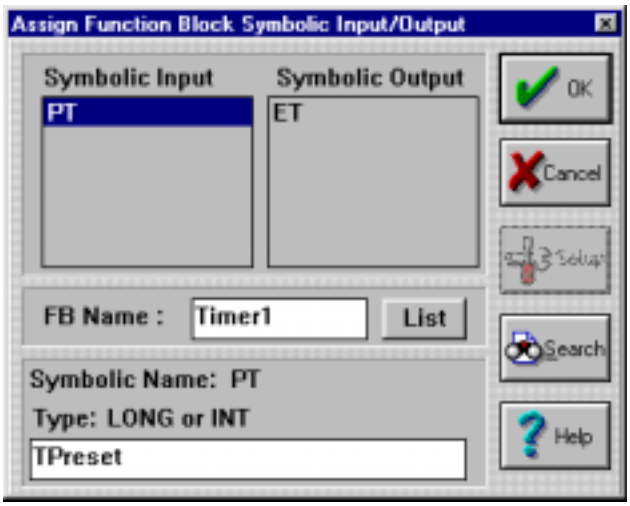


Figure 10.2 Assign Function Block Symbolic Input/Output Dialog Box

Counter Down (CTD)

The Counter Down function block, shown in *Figure 10.3 on page 118*, enables an output after a preset number of transitions from False to True occur.

CTD decrements the integer current value (CV) for every False to True transition of the counter down enable bit (CD). The current value continues to decrement until it reaches zero. At this point, the output bit (Q) is enabled and remains enabled until the function block is reset by a True condition of the load bit (LD). When the load bit is True, the current value resets and remains at the integer preset value (PV).

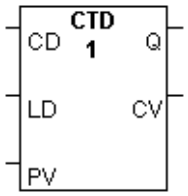


Figure 10.3 Counter Down Function Block

Inputs

CD is the Counter Down enable bit. As long as the reset bit is False, each False to True transition of this bit decrements the current value until the current value equals zero. In a program using Ladder logic, the value of CD is determined by the Boolean evaluation of elements placed between the left power rail and CD in the function block. In FBD programs, the value of CD is determined by the Variable connected to CD. If no contacts are placed on the rung before or Variable attached to CD, the value of CD is always True, but a transition never occurs.

LD is the Counter Down load bit. When this bit is True, the current value is set to the preset value and the output bit is disabled. In Ladder Diagram programs, the value of LD is determined by the Boolean evaluation of elements placed between the left power rail and LD. In FBD programs, the value of LD is determined by the Variable connected to LD.

PV is the Counter Down preset value. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from 1 to 32,767 for **INT** type Variables and 1 to 2,147,483,647 for **LONG** type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

Q is the Counter Down output bit. This bit is enabled and passes power when the current value is equal to zero. Prior to an initial loading of the preset value, Q is not enabled even though the current value is set to zero.

CV is the Counter Down current value. In Ladder logic, assign this output a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this output in the **Variable Assignment** dialog box. The value of this Variable

decrements by one every time a False to True transition of the counter down enable bit occurs until the current value equals zero. The current value retains this value until the reset bit is True. Prior to an initial loading of the preset value, CV has a value of zero.

Counter Up Down (CTUD)

The Counter Up Down function block, shown in *Figure 10.4*, is a combination of the counter up and counter down function blocks.

CTUD increments the integer current value (CV) for every False to True transition of the counter up enable bit (CU), and decrements the current value for every False to True transition of the counter down enable bit (CD). The current value continues to increment or decrement depending on which enable bit becomes True until either the integer preset value (PV) or zero is reached. At this point, the up output bit (QU) is enabled if the current value equals the preset value, or the down output bit (QD) is enabled if the current value equals zero.

The output bit remains enabled until reset by a True condition of the (R) or (LD) bit respectively. When the reset bit (R) is True, the current value resets and remains at zero. When the load bit (LD) is True, the current value resets and remains at the preset value.

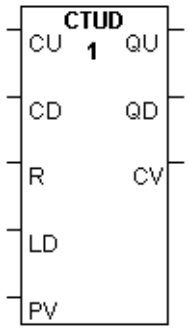


Figure 10.4 Counter Up Down Function Block

Inputs

CU is the Counter Up enable bit. As long as the reset bit is False, each False to True transition of this bit increments the current value until the current value equals the preset value. In a program using Ladder logic, the value of CU is determined by the Boolean evaluation of elements placed between the left power rail and CU in the function block. In FBD programs, the

value of CU is determined by the Variable connected to CU. If no contacts are placed on the rung before CU, the value of CU is always True, but a transition never occurs. This bit has higher evaluation priority than the CD bit in this function block.

CD is the Counter Down enable bit. As long as the reset bit is False, each False to True transition of this bit decrements the current value until the current value equals zero. In a Ladder Diagram program, the value of CD is determined by the Boolean evaluation of elements placed between the left power rail and CD in the function block. In FBD programs, the value of CD is determined by the Variable connected to CD. If no contacts are placed on the rung before CD, the default value of CD is True. This bit has the lowest evaluation priority of the input bits in this function block.

R is the Counter Up reset bit. When this bit is True, the current value is set to zero and the QD output bit is disabled. In a Ladder Diagram program, the value of R is determined by the Boolean evaluation of elements placed between the left power rail and R in the function block. In FBD programs, the value of R is determined by the Variable connected to R. If no contacts are placed on the rung before R, the value of R is always False. This bit has the highest evaluation priority of the input bits in this function block.

LD is the Counter Down load bit. When this bit is True, the current value is set to the preset value and the QU output bit is disabled. In a Ladder Diagram program, the value of LD is determined by the Boolean evaluation of elements placed between the left power rail and LD in the function block. In FBD programs, the value of LD is determined by the Variable connected to LD. If no contacts are placed on the rung before LD, the value of LD is always False. This bit has higher evaluation priority than the CD bit in this function block.

PV is the Counter Up Down preset value. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from 1 to 32,767 for **INT** type Variables and 1 to 2,147,483,647 for **LONG** type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

QU is the Counter Up output bit. When CU passes power, this bit is enabled and passes power once the current value is equal to the preset value if CV counted up to PV (not if LD was used to load CV with the PV value).

QD is the Counter Down output bit. This bit is enabled and passes power when the current value is equal to zero if CV counted down to zero (not if "reset" was used to reset CV to zero). A rung with an output coil is not needed between QD in the function block and the right power rail or connect an output Variable to it. Prior to an initial loading of the preset value, QD is not enabled even though the current value is set to zero.

CV is the Counter Up Down current value. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this output in the **Variable Assignment** dialog box. The value of this Variable increments by one every time a False to True transition of the counter up enable bit occurs, and decrements by one every time a False to True transition of the counter down enable bit occurs, until the current value equals either the preset value or zero. The current value retains this value until the reset bit is True. Prior to an initial loading of the preset value, the value of CV is zero.

Timer Pulse (TP)

The Timer Pulse function block, shown in *Figure 10.5*, enables an output for a preset amount of time.

Upon a True condition of the enable bit (INP), TP increments the integer elapsed time (ET) value in milliseconds until the integer preset time (PT) value is reached. During this period of time only, the output bit (Q) is enabled. The elapsed time value resets to zero once it reaches the preset value and the enable bit becomes False.

NOTE! ***Once the timer starts, status of INP does not effect the count.***

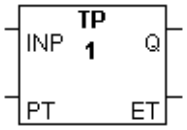


Figure 10.5 Timer Pulse Function Block

Inputs

INP is the Timer Pulse enable bit. A True condition of this bit triggers the elapsed time to start incrementing in thousandths of a second until the elapsed time equals the preset time. In a program using Ladder logic, the value of INP is determined by the Boolean evaluation of elements placed between the left power rail and INP in the function block. In FBD programs, the value of INP is determined by the Variable connected to INP. If no contacts are placed on the rung before INP, the value of INP is always True.

PT is the Timer Pulse preset time. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from 1 to 32,767 for **INT** type Variables and 1 to 2,147,483,647 for **LONG** type Variables thousandths of a second. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

Q is the Timer Pulse output bit. This bit is enabled and passes power when the elapsed time is less than the preset time and is still incrementing.

ET is the Timer Pulse elapsed time. In ladder logic, this Variable may be assigned a symbolic name in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this output in the **Variable Assignment** dialog box. The value of this Variable increments in thousandths of a second when triggered by a True condition of the enable bit until the elapsed time is equal to the preset time. The elapsed time retains this value until the enable bit is False. If the enable bit becomes False before the elapsed time equals the preset time, the elapsed time continues to increment, but resets to zero as soon as it reaches the preset value.

Figure 10.6 on page 123 illustrates the relationship between the inputs and outputs of the TP function block and shows how the INP signal and ET (timer) determine the state of the output Q. INP and Q signals are high or low, the ET is an analog value on the vertical axis. Time is on the horizontal axis.

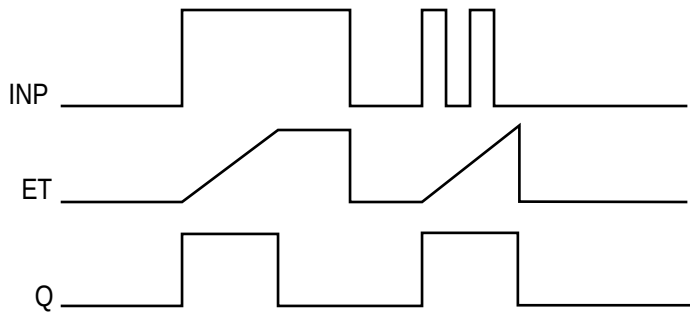


Figure 10.6 Relationship between the Input and Output of TP Function Block

Timer On-Delay (TON)

The Timer On-Delay function block, shown in *Figure 10.7*, provides a time delay for enabling an event at a preset time interval.

Upon a True condition of the enable bit (INP), TON increments the integer elapsed time (ET) value in thousandths of a second until the integer preset time (PT) value is reached. Once the elapsed time value equals the preset time value, the output bit (Q) is enabled and remains enabled until the enable bit becomes False. The elapsed time value resets to and remains at zero as long as the enable bit is False.

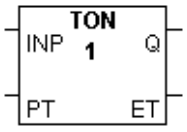


Figure 10.7 Timer On-Delay Function Block

Inputs

INP is the Timer On-Delay enable bit. A True condition of this bit triggers the elapsed time to start incrementing in thousandths of a second until the elapsed time equals the preset time. After this delay, the output bit is enabled as long as the enable bit remains True. In a program using Ladder logic, the value of INP is determined by the Boolean evaluation of elements placed between the left power rail and INP in the function block. In FBD programs, the value of INP is determined by the Variable connected to INP. If no contacts are placed on the rung before INP, the value of INP is always True.

PT is the timer on-delay preset time. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from 1 to 32,767 for **INT** type Variables and 1 to 2,147,483,647 for **LONG** type Variables thousandths of a second. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

Q is the timer on-delay output bit. This bit is enabled and passes power when the elapsed time becomes equal to the preset time, and remains enabled until the enable bit is False.

ET is the timer on-delay elapsed time. The value of this Variable increments in thousandths of a second when triggered by a True condition of the enable bit until the elapsed time is equal to the preset time. The elapsed time retains this value until the enable bit is False. If the enable bit becomes False before the elapsed time equals the preset time, the elapsed time resets to zero.

Figure 10.8 illustrates the relationship between the inputs and outputs of the TON function block and shows how the INP signal and ET (timer) determine the state of the output Q, INP and Q signals are high or low. ET is an analog value on the vertical axis. Time is on the horizontal axis.

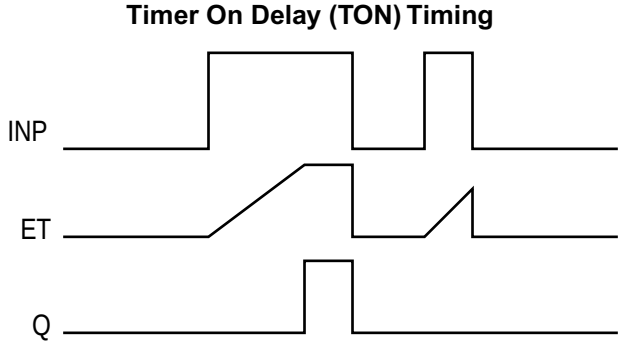


Figure 10.8 Relationship Between Inputs and Outputs of TON Function Block

Timer Off-Delay (TOF)

The Timer Off-Delay function block, shown in *Figure 10.9 on page 125*, provides a time delay for disabling an event at a preset time interval.

The output bit (Q) is enabled upon a True condition of the enable bit (INP). When there is a False condition of the enable bit, TOF continues to enable the output bit while it increments the integer elapsed time (ET) value in thousandths of a second until the integer preset time (PT) value is reached. Once the elapsed time value equals the preset time value, the output is disabled. The elapsed time value resets and remains at zero as long as the enable bit is False.

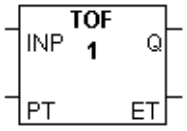


Figure 10.9 Timer Off-Delay Function Block

Inputs

INP is the Timer Off-Delay enable bit. A True condition of this bit enables the output bit. A False condition of this bit triggers the elapsed time to start incrementing in thousandths of a second until the elapsed time equals the preset time. After this delay, the output bit is disabled until the next True condition of the enable bit. In a program using Ladder logic, the value of INP is determined by the Boolean evaluation of elements placed between the left power rail and INP in the function block. In an FBD program, it is determined by the Variable connected to INP. If no contacts are placed on the rung before INP, the value of INP is always True.

PT is the Timer Off-Delay preset time. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from 1 to 32,767 for **INT** type Variables and 1 to 2,147,483,647 for **LONG** type Variables thousandths of a second. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

Q is the timer off-delay output bit. This bit is enabled and passes power when the enable bit is True. When the enable bit is False, it remains enabled until the elapsed time becomes equal to the preset time.

ET is the timer off-delay elapsed time. The value of this Variable increments in thousandths of a second when triggered by a False condition of the enable bit until the

elapsed time is equal to the preset time. The elapsed time retains this value until the enable bit is True again. Prior to the first initialization of this function block, the elapsed time does not increment even though the enable bit is False.

Figure 10.10 illustrates the relationship between the inputs and outputs of the TOF function block and shows how the INP signal and ET (timer) determine the state of the output Q, INP and Q signals are high or low. ET is an analog value on the vertical axis. Time is on the horizontal axis.

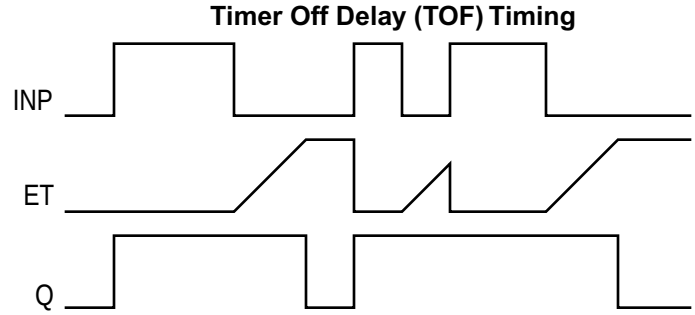


Figure 10.10 Relationship Between Inputs and Outputs of TOF Function Block

Comparison

The following section contains descriptions of comparison function blocks found in the *standard* category.

Greater Than (GT)

The Greater Than (GT) function block, shown in *Figure 10.11*, tests if a value is greater than a reference value.

GT tests whether one value (A) is greater than another value (B) when the enable bit (EN) is True. If the relationship A>B is True, then the output bit (Q) is enabled. The output bit is disabled either when the enable bit is False or the value of A is no longer greater than B. The values of A and B can be preset to a constant or assigned a symbolic name, which allows their value to change dynamically within the program.

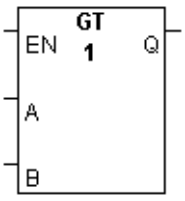


Figure 10.11 Greater Than Function Block

Inputs

EN is the Greater Than enable bit. A True condition of this bit causes the comparison between A and B to occur. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN, the value of EN is always True.

A is one comparison value. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

B is another comparison value. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Output

Q is the Greater Than output bit. This bit is enabled and passes power when the enable bit is True and the value of A is greater than the value of B.

Less Than (LT)

The Less Than function block, shown in *Figure 10.12 on page 128*, tests if a value is less than a reference value.

LT tests whether one value (A) is less than another value (B) when the enable bit (EN) is True. If the relationship A<B is True, then the output bit (Q) is enabled. The output bit is disabled either when the enable bit becomes False or the value of A is no longer less than B. The values of A and B can be preset to a constant or assigned a symbolic name, which allows their value to change dynamically within the program.

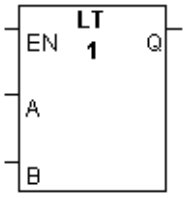


Figure 10.12 Less Than Function Block

Inputs

EN is the Less Than (LT) enable bit. A True condition of this bit causes the comparison between A and B to occur. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN, the value of EN is always True.

A is one comparison value. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

B is another comparison value. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Output

Q is the Less Than output bit. This bit is enabled and passes power when the enable bit is True and the value of A is less than the value of B.

Equal To (EQ)

The Equal To (EQ) function block, shown in *Figure 10.13 on page 129*, tests if a value is equal to a reference value.

EQ tests whether one value (A) is equal to another value (B) when the enable bit (EN) is True. If the relationship A=B is True, then the output bit (Q) is enabled. The output bit is disabled either when the enable bit is False or the value of A is no longer equal to B. The values of A and B can be preset to a constant or assigned a symbolic name, which allows their value to change dynamically within the program.

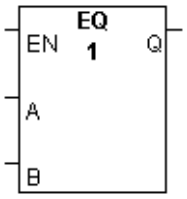


Figure 10.13 Equal To Function Block

Inputs

EN is the Equal To enable bit. A True condition of this bit causes the comparison between A and B to occur. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN, the value of EN is always True.

A is one comparison value. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

B is another comparison value. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Output

Q is the Equal To output bit. This bit is enabled and passes power when the enable bit is True and the value of A is equal to the value of B.

Arithmetic

The following sections contain descriptions of function blocks found in the *standard* category that perform arithmetic.

Add (ADD)

The Add (ADD) function block, shown in *Figure 10.14*, produces the sum of two values.

ADD calculates the sum of one value (A) and another value (B) and places the result in another Variable (C), when the enable bit (EN) is True. The values of A and B can be preset to a constant or assigned a Variable name, which allows their value to change dynamically within the program.

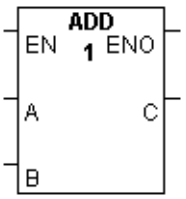


Figure 10.14 ADD Function Block

Inputs

EN is the ADD enable bit. A True condition of this bit causes the value of A to be added to the value of B and the result stored in C. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in Ladder programs. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in Ladder programs, the value of EN is always True.

A is one value to be added. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

B is another value to be added. The range of values this Variable can assume is from -32,768 to 32,767, including 0

for **INT** type Variables and -2,147,483,648 to 2,147,483,647, including 0 for **LONG** type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

ENO is the Add enable out bit. This bit is enabled and passes power when the enable bit is True, as long as the value of C does not go out of range.

C is the ADD sum. C contains the sum of A and B, as long as the enable bit is True. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, ENO is disabled. C reverts to its last valid value whenever ENO is disabled.

Subtract (SUB)

The Subtract (SUB) function block, shown in *Figure 10.15*, produces the difference between two values.

SUB calculates the difference between one value (A) and another value (B) and places the result in another Variable (C), when the enable bit (EN) is True. The values of A and B can be preset to a constant or assigned a symbolic name, which allows their value to change dynamically within the program.

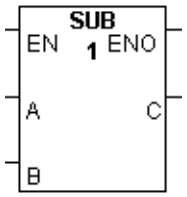


Figure 10.15 Subtract Function Block

Inputs

EN is the Subtract enable bit. A True condition of this bit causes the value of B to be subtracted from the value of A and the result stored in C. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder program, the value

of EN is always True. In FBD, all inputs must be connected to a Variable.

A is the value to be subtracted from. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

B is the value to subtract. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

ENO is the Subtract enable out bit. This bit is enabled and passes power when the enable bit is True, as long as the value of C does not go out of range.

C is the difference. C contains the difference between A and B, as long as the enable bit is True. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables.

Multiply (MULT)

The Multiply (MULT) function block, shown in *Figure 10.16*, produces the product of two values.

MULT calculates the product between one value (A) and another value (B) and places the result in another Variable (C), when the enable bit (EN) is True. The values of A and B can be preset to a constant or assigned a Variable name, which allows their value to change dynamically within the program.

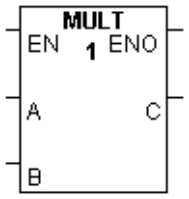


Figure 10.16 Multiply Function Block

Inputs

EN is the Multiply enable bit. A True condition of this bit causes the values of A and B to be multiplied and the result stored in C. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

A is a multiply value. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

B is another multiply value. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

ENO is the Multiply enable out bit. This bit is enabled and passes power when the enable bit is True, as long as the value of C does not go out of range.

C is the product value. C contains the product of A and B, as long as the enable bit is True. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for **INT** type Variables and -2,147,483,648 to 2,147,483,647, including 0 for **LONG** type Variables. If this Variable is assigned a value out of these ranges, ENO is disabled. C reverts to its last valid value whenever ENO is disabled.

Divide (DIV)

The Divide (DIV) function block, shown in *Figure 10.17 on page 134*, produces the quotient of two values.

DIV calculates the quotient between one value (A) which is the dividend and another value (B) which is the divisor and places the result in another Variable (C), when the enable bit (EN) is True. The values of A and B can be preset to a constant or assigned a Variable name, which allows their value to change dynamically within the program.

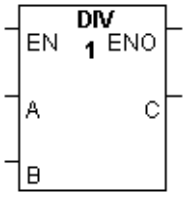


Figure 10.17 Divide Function Block

Inputs

EN is the Divide enable bit. A True condition of this bit causes the value of A to be divided by the value of B and the result stored in C. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagrams program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

A is the dividend. The range of values this Variable can assume is from -32,768 to 32,767, excluding 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, excluding 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

B is the divisor. The range of values this Variable can assume is from -32,768 to 32,767, excluding 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, excluding 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur. Also, if B is assigned a value of zero (0), the enable out bit goes False and C retains its previous value.

NOTE! ***If the divisor is 0, an error occurs as per standard mathematical convention.***

Outputs

ENO is the Divide enable out bit. This bit is enabled and passes power when the enable bit is True as long as the value of B is non-zero and value of C does not go out of range.

C is the quotient. C contains the quotient of A and B, as long as the enable bit is True. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. No remainders are stored in this value; only the whole number value. If this Variable is

assigned a value out of these ranges, ENO is disabled. C reverts to its last valid value whenever ENO is disabled.

Data Manipulation

The following section contains a description of the data manipulation function blocks found in the *standard* category.

First In - First Out (FIFO)

The First In - First Out (FIFO) function block, shown in *Figure 10.18*, creates and manipulates a stack of integers.

FIFO moves the contents of the input Variable (INP) to the stack when it encounters a False to True transition of the load bit (LD). The size of the stack is specified by assigning a constant to the SIZE parameter. The count Variable (CNT) tracks the number of integers in the stack.

When the stack is filled with integers, the full bit (FUL) becomes True. This bit only remains True as long as the count of integers in the stack is equal to the size of the stack. A False to True transition of the unload bit (UNLD) moves the first integer entered in the stack to the output Variable (OUTP) and changes the output bit (Q) to True. Every succeeding False to True transition of the unload bit moves the oldest integer from the stack to the output Variable and causes the output bit to remain True until the entire stack is unloaded.

At this point, the count Variable equals zero, the output bit is False, and the empty bit (EMP) is True. This bit only remains True as long as the count of integers in the stack is equal to zero. A True condition of the clear bit (CLR) clears the stack of any integer values and resets the count Variable to zero, thus causing the empty bit to be True. The output Variable retains its last value when the stack is cleared.

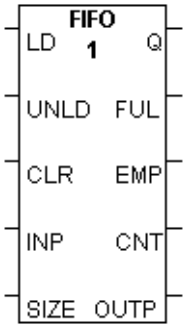


Figure 10.18 First In - First Out Function Block

Inputs

LD is the FIFO load bit. As long as the clear bit is False, each False to True transition of this bit loads the input Variable value into the stack until the count value equals the size value. In a program using Ladder logic, the value of LD is determined by the Boolean evaluation of elements placed between the left power rail and LD in the function block. In FBD programs, the value of LD is determined by the Variable connected to LD. If no contacts are placed on the rung before LD, the value of LD is always True, but a transition never occurs. This bit has the lowest evaluation priority of the input bits in this function block.

UNLD is the FIFO unload bit. As long as the clear bit is False and the count value is not zero, each False to True transition of this bit unloads the oldest integer in the stack to the output Variable. In a Ladder Diagram program, the value of UNLD is determined by the Boolean evaluation of elements placed between the left power rail and UNLD in the function block. In FBD programs, the value of UNLD is determined by the Variable connected to UNLD. If no contacts are placed on the rung before UNLD, the value of UNLD is always False. This bit has higher evaluation priority than the LD bit in this function block.

CLR is the FIFO clear bit. When this bit is True, the stack is cleared of any values and the count Variable is set to zero. In a Ladder Diagram program, the value of CLR is determined by the Boolean evaluation of elements placed between the left power rail and CLR in the function block. In FBD programs, the value of CLR is determined by the Variable connected to EN. If no contacts are placed on the rung before CLR, the value of CLR is always False. This bit takes the highest evaluation priority of the input bits in this function block.

INP is the FIFO input Variable. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The value of this Variable is placed in the stack at the location of the count Variable. The range values this Variable can assume is from -32,768 to 32,767, including 0. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

SIZE is the FIFO size Variable. This value assigned to this Variable determines the size of the stack. For this reason, assign a constant value and do not change this value online.

Outputs

Q is the FIFO output bit. This bit is enabled and passes power when a False to True transition of the unload bit occurs. This bit remains enabled until there are no more values in the stack to unload and the count Variable equals zero.

FUL is the FIFO full bit. This bit is enabled and passes power when the count value is equal to the size value. There does not have to be a rung with an output coil placed between FUL in the function block and the right power rail (in a Ladder Diagram program) or an output Variable connected to it (in a FBD program).

EMP is the FIFO empty bit. This bit is enabled and passes power when the count value equals zero. A rung with an output coil is not needed between EMP in the function block and the right power rail (in a Ladder Diagram program) or an output Variable connected to it (in a FBD program).

CNT is the FIFO count Variable. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this output in the **Variable Assignment** dialog box. The value of this Variable indicates the number of integers in the stack.

OUTP is the FIFO output Variable. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this output in the **Variable Assignment** dialog box. This Variable holds the value unloaded from the stack. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable exceeds these ranges, unpredictable results will occur.

Last In - First Out (LIFO)

The Last In - First Out (LIFO) function block, shown in *Figure 10.19 on page 138*, creates and manipulates a stack of integers.

LIFO moves the contents of the input Variable (INP) to the stack when it encounters a False to True transition of the load bit (LD). The size of the stack is specified by assigning a constant or symbolic Variable to the SIZE parameter. The count Variable (CNT) tracks the number of integers in the stack.

When the stack is filled with integers, the full bit (FUL) is True. This bit only remains True as long as the count of

integers in the stack is equal to the size of the stack. A False to True transition of the unload bit (UNLD) moves the last integer entered in the stack to the output Variable (OUTP) and causes the output bit (Q) to be True. Every succeeding False to True transition of the unload bit continues to move the newest integer from the stack to the output Variable, and causes the output bit to remain True until the entire stack is unloaded.

At this point, the count Variable equals zero, the output bit is False, and the empty bit (EMP) is True. This bit only remains True as long as the count of integers in the stack is equal to zero. A True condition of the clear bit (CLR) clears the stack of any integer values and resets the count Variable to zero, thus causing the empty bit to be True. The output Variable retains its last value when the stack is cleared.

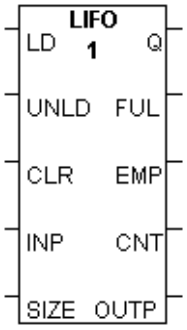


Figure 10.19 Last In - First Out Function Block

Inputs

LD is the LIFO load bit. As long as the clear bit is False, each False to True transition of this bit loads the input Variable value into the stack until the count value equals the size value. In a program using Ladder logic, the value of LD is determined by the Boolean evaluation of elements placed between the left power rail and LD in the function block. In FBD programs, the value of LD is determined by the Variable connected to LD. If no contacts are placed on the rung before LD, the value of LD is always True, but a transition never occurs. This bit has the lowest evaluation priority of the input bits in this function block.

UNLD is the LIFO unload bit. As long as the clear bit is False and the count value is not zero, each False to True transition of this bit unloads the newest integer in the stack to the output Variable. In a Ladder Diagram program, the value of UNLD is determined by the evaluation of Boolean elements placed between the left power rail and UNLD in the function block. In

FBD programs, the value of UNLD is determined by the Variable connected to UNLD. If no contacts are placed on the rung before UNLD, the value of UNLD is always False. This bit has higher evaluation priority than the LD bit in this function block.

CLR is the LIFO clear bit. When this bit is True, the stack is cleared of any values and the count Variable is set to zero. In a Ladder Diagram program, the value of CLR is determined by the Boolean evaluation of elements placed between the left power rail and CLR in the function block. In FBD programs, the value of CLR is determined by the Variable connected to CLR. If no contacts are placed on the rung before CLR, the value of CLR is always False. This bit takes the highest evaluation priority of the input bits in this function block.

INP is the LIFO input Variable. In Ladder logic, assign this input a symbolic name or constant value in the **Assign Function Block Symbolic Input/Output** dialog box. In FBD, assign a variable to this input in the **Variable Assignment** dialog box. The value of this Variable is placed in the stack at the location of the count Variable. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

SIZE is the LIFO size Variable. The value assigned to this Variable determines the size of the stack. For this reason, assign a constant value and do not change this value online. This Variable should be either a positive integer or a constant.

Outputs

Q is the LIFO output bit. This bit is enabled and passes power when a False to True transition of the unload bit occurs. This bit remains enabled until there are no more values in the stack to unload and the count Variable equals zero.

FUL is the LIFO full bit. This bit is enabled and passes power when the count value is equal to the size value. A rung with an output coil is not needed between FUL in the function block and the right power rail (in a Ladder Diagram program) or connect an output Variable to it (in a FBD program).

EMP is the LIFO empty bit. This bit is enabled and passes power when the count value equals zero. A rung with an output coil is not needed between EMP in the function block and the right power rail (in a Ladder Diagram program) or connect an output Variable to it (in a FBD program).

CNT is the LIFO count Variable. The value of this Variable indicates the number of integers in the stack.

OUTP is the LIFO output Variable. This Variable holds the value unloaded from the stack. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables. If this Variable is exceeds these ranges, unpredictable results will occur.

Bit Shift (SL)

The Bit Shift (SL) function block, shown in *Figure 10.20*, creates and manipulates a synchronous bit shift register.

SL moves the contents of the shift register one bit to the left upon a False to True transition of the shift bit (SHFT). The value of the input bit (BIT) then is the first bit in the shift register. The size of the shift register is determined by the integer value of SIZE. Once the register is filled by shifting in the number of bits equal to the size of the register, the next False to True transition of the shift bit causes the last bit of the register to shift into the output bit (Q). A True condition of the clear bit (CLR) clears the shift register.

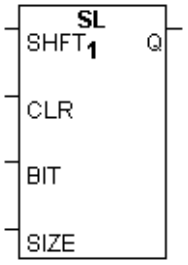


Figure 10.20 Bit Shift Function Block

Inputs

SHFT is the SL shift bit. As long as the clear bit is False, each False to True transition of this bit shifts the contents of the register one position to the left in the register. The value of the input bit fills in the first position of the register and the last value in the register is placed in the output bit. In a program using Ladder logic, the value of SHFT is determined by the Boolean evaluation of elements placed between the left power rail and SHFT in the function block. In FBD programs, the value of SHFT is determined by the Variable connected to SHFT. If no contacts are placed on the rung before SHFT, the default value of SHFT is True; however nothing can ever be shifted since there is never a transition. The SHFT bit has the

lowest evaluation priority of the input bits in this function block.

CLR is the SL clear bit. When this bit is True, the register is cleared of all values. In a Ladder Diagram program, the value of CLR is determined by the Boolean evaluation of elements placed between the left power rail and CLR in the function block. In FBD programs, the value of CLR is determined by the Variable connected to CLR. If no contacts are placed on the rung before CLR, the default value of CLR is False. This bit takes the highest evaluation priority of the inputs in this function block.

BIT is the SL input bit. This value of this bit is placed in the first position in the register when a False to True transition of the shift bit occurs, as long as the clear bit is False. In a Ladder Diagram program, the value of BIT is determined by the Boolean evaluation of elements placed between the left power rail and BIT in the function block. In FBD programs, the value of BIT is determined by the Variable connected to BIT. If no contacts are placed on the rung before BIT, the default value of BIT is False. In this scenario, only zeroes are shifted into the register.

SIZE is the SL size Variable. The value assigned to this Variable determines the size of the stack. For this reason, assign a constant value and do not change this value online. This Variable should be either a positive INT or LONG Variable or a constant.

Outputs

Q is the SL output bit. This bit is enabled and passes power when a bit with a True value is shifted out of the register. The output bit remains enabled until a bit with a False value is shifted out of the register or the clear bit is True. When a bit with a False value is shifted out of the register, the output bit does not pass power.

Figure 10.21 illustrates how bits are shifted through the SL function block.

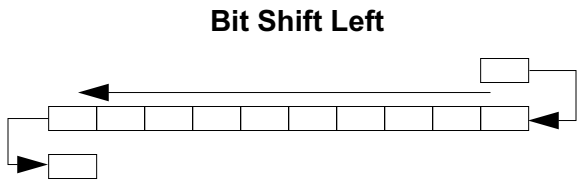


Figure 10.21 Example of How Bits Shift Through the SL Function Block

Scan Time Counter (SCAN)

The Scan Time Counter (SCAN) function block, shown in *Figure 10.22*, calculates the scan time system performance while running a program.

SCAN calculates the scan time when the enable bit (EN) is True. The current scan time (TIME) is displayed along with the minimum value (MIN) and the maximum value (MAX) in thousandths of a second. The output bit (Q) always assumes the value of the enable bit; therefore, when the enable bit is True, the output bit is also True.

The symbolic Variables for scan time, minimum, and maximum must be of type Integer. These values change dynamically as the program runs, as long as the enable bit is True. When the enable bit becomes False, the Variables retain their last value. The clear bit (CLR) resets and holds the Variables at zero whenever it is True.

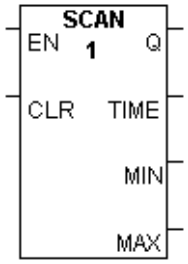


Figure 10.22 Scan Time Counter Function Block

Inputs

EN is the Scan Time Counter enable bit. A True condition of this bit causes the scan time to be calculated. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagrams program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

CLR is the Scan Time Counter clear bit. When this bit is True, the values of TIME, MIN, and MAX are set and remain at zero. In a Ladder Diagram program, the value of CLR is determined by the Boolean evaluation of elements placed between the left power rail and CLR in the function block. In FBD programs, the value of CLR is determined by whatever is connected to EN. If no contacts are placed on the rung before CLR in a

Ladder Diagrams program, the value of CLR is always False. In FBD, all inputs must be connected to a Variable.

Outputs

Q is the Scan Time Counter output bit. This bit is enabled and passes power while a True condition of the enable bit occurs.

TIME is the scan time Variable. The value of this Variable indicates the scan time in thousandths of a second.

MIN is the minimum scan time Variable. The value of this Variable indicates the minimum scan time in thousandths of a second since the program started running.

MAX is the maximum scan time Variable. The value of this Variable indicates the maximum scan time in thousandths of a second since the program started running.

Move (MOVE)

The Move (MOVE) function block, shown in *Figure 10.23*, places the value of one Variable into another Variable.

MOVE places the input Variable (INP) value in the output Variable (OUTP) while the enable bit (EN) is True. The output bit (Q) always assumes the value of the enable bit; therefore, when the enable bit is True, the output bit is also True. The symbolic Variables for INP and OUTP must be either INT or LONG. When the enable bit is False, the OUTP Variable retains its last value.

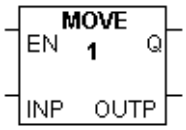


Figure 10.23 Move Function Block

Inputs

EN is the MOVE enable bit. A True condition of this bit copies the value of INP to OUTP. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagrams program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

INP is the MOVE input Variable. The range of values this Variable can assume is from -32,768 to 32,767, including 0 for INT type Variables and -2,147,483,648 to 2,147,483,647, including 0 for LONG type Variables.

Output

Q is the MOVE output bit. This bit is enabled and passes power whenever the enable bit is True.

OUTP is the move output Variable. The value of the input Variable is copied to this Variable when the enable bit is True. OUTP reverts to its last valid value whenever EN is disabled.

Scale (SCL)

The Scale (SCL) function block, shown in *Figure 10.24*, scales an INT value of 0 to 4095 to a REAL value based on RMAX and RMIN parameters.

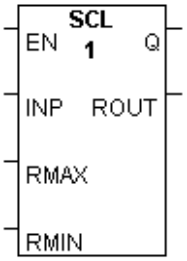


Figure 10.24 Scale Function Block

Inputs

EN is an enable bit and has a Boolean value. If the bit is False, the evaluation of the SCL is skipped. If the bit is True, the SCL function block is evaluated.

INP is an INT constant or Variable from 0 to 4095 which will be scaled to a REAL based on RMAX and RMIN range settings. The absolute range of values this Variable can assume is from -32,768 to 32,767, including 0. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Both RMAX and RMIN scale INT INP into a ROUT. RMAX and RMIN are of type REAL. The range of values this Variable can assume is from -3.4×10^{38} to 3.4×10^{38}. If this Variable is assigned a value out of these ranges, unpredictable results will occur. ROUT is equal to RMIN when INP is 0. ROUT is equal to RMAX when INP is 4095.

Outputs

Q is the Boolean output bit.

ROUT is the REAL result of the SCL operation. The value is based on the settings for RMAX and RMIN. ROUT retains its previous value whenever EN is disabled. See *Figure 10.25*.

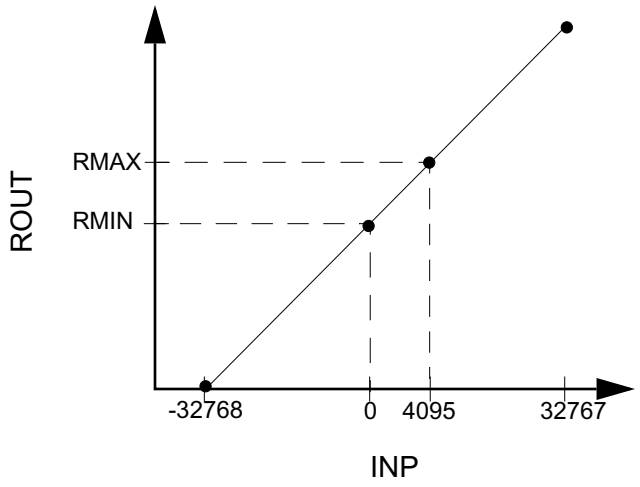


Figure 10.25 ROUT as a function of INP

Real Number Function Blocks

Real function blocks are similar to regular standard function blocks in *LogicPro*. As the name implies, real function blocks use real Variables to perform calculations. In turn, the output of these function blocks is also real. These function blocks are found in the *standard* category.

Real Greater Than (RGT)

The Real Greater Than (RGT) function block, as shown in *Figure 10.26 on page 146*, tests if a real Variable value is greater than a reference value.

Like the Greater Than function block, RGT tests whether one value (RA) is greater than another value (RB) when the enable bit (EN) is True. If the relationship RA>RB is True, then the output bit (Q) is enabled. The output bit is disabled either when the enable bit becomes False or the value of RA is no longer greater than RB. The values of A and B can be preset to a constant or assigned a symbolic name, which allows their value to change dynamically within the program.

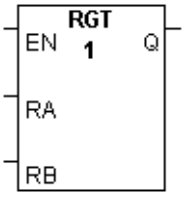


Figure 10.26 Real Greater Than Function Block

Inputs

EN is the Real Greater Than enable bit. A True condition of this bit causes the comparison between RA and RB to occur. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of CU is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagrams program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

RA is one comparison value. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

RB is another comparison value. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Output

Q is the Real Greater Than output bit. This bit is enabled and passes power when the enable bit is True and the value of RA is greater than the value of RB.

Real Less Than (RLT)

The Real Less Than (RLT) function block, as shown in *Figure 10.27 on page 147*, tests if a real Variable value is less than a reference value.

RLT tests whether one value (RA) is less than another value (RB) when the enable bit (EN) is True. If the relationship RA<RB is True, then the output bit (Q) is enabled. The output bit is disabled either when the enable bit becomes False or the value of RA is no longer less than RB. The values of RA and RB can be preset to a constant or assigned a symbolic name, which allows their value to change dynamically within the program.

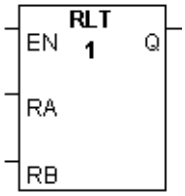


Figure 10.27 Real Less Than Function Block

Inputs

EN is the Real Less Than enable bit. A True condition of this bit causes the comparison between RA and RB to occur. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

RA is one comparison value. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

RB is another comparison value. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Output

Q is the Real Less Than output bit. This bit is enabled and passes power when the enable bit is True and the value of RA is less than the value of RB.

Real Equal To (REQ)

The Real Equal To (REQ) function block, as shown in *Figure 10.28 on page 148*, tests if a real Variable value is equal to a reference value.

REQ tests whether one value (RA) is equal to another value (RB) when the enable bit (EN) is True. If the relationship RA=RB is True, then the output bit (Q) is enabled. The output bit is disabled either when the enable bit becomes False or the value of RA is no longer equal to RB. The values of RA and RB can be preset to a constant or assigned a symbolic name,

which allows their value to change dynamically within the program.

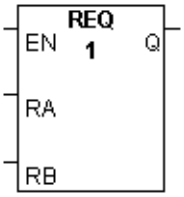


Figure 10.28 Real Equal To Function Block

Inputs

EN is the Real Equal To enable bit. A True condition of this bit causes the comparison between RA and RB to occur. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagrams program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

RA is one comparison value. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

RB is another comparison value. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Output

Q is the Real Equal To output bit. This bit is enabled and passes power when the enable bit is True and the value of A is equal to the value of B.

Real Add (RADD)

The Real Add (RADD) function block, shown in *Figure 10.29 on page 149*, produces the sum of two real values.

RADD calculates the sum of one value (RA) and another value (RB) and places the result in another real Variable (RC), when the enable bit (EN) is True. The values of RA and RB can be preset to a constant or assigned a Variable name, which allows their value to change dynamically within the program.

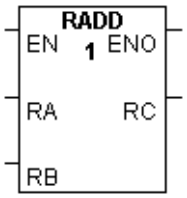


Figure 10.29 Real Add Function Block

Inputs

EN is the Real Add enable bit. A True condition of this bit causes the value of RA to be added to the value of RB and the result stored in RC. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagram program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

RA is one value to be added. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

RB is another value to be added. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

ENO is the Real Add output bit. This bit is enabled and passes power when the enable bit EN is True and the value of RC does not go out of range.

RC is the Real Add result value. RC contains the sum of RA and RB, as long as the enable bit is True. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. This Variable retains its previous value if EN is not True. Likewise, if the value of RC falls outside this range, ENO is disabled. RC reverts to its last valid value whenever ENO is disabled.

Real Subtract (RSUB)

The Real Subtract (RSUB) function block, shown in *Figure 10.30 on page 150*, produces the difference between two values.

RSUB calculates the difference between one value (RA) and another value (RB) and places the result in another Variable (RC) when the enable bit (EN) is True. The values of RA and RB can be preset to a constant or assigned a Variable name, which allows their value to change dynamically within the program.

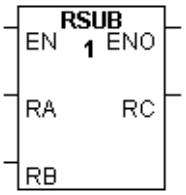


Figure 10.30 Real Subtract Function Block

Inputs

EN is the Real Subtract enable bit. A True condition of this bit causes the value of RB to be subtracted from the value of RA and the result stored in RC. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagram program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

RA is the value to be subtracted from. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

RB is the value to subtract. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

ENO is the Real Subtract enable out bit. This bit is enabled and passes power when the enable bit is True as long as the value of RC does not go out of range.

RC is the subtract result value. RC contains the difference between RA and RB, as long as the enable bit is True. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, ENO is disabled. RC reverts to its last valid value whenever ENO is disabled.

Real Multiply (RMUL)

The Real Multiply (RMUL) function block, shown in *Figure 10.31*, produces the product of two values.

RMUL calculates the product between one value (RA) and another value (RB) and places the result in another Variable (RC) when the enable bit (EN) is True. The values of RA and RB can be preset to a constant or assigned a Variable name, which allows their value to change dynamically within the program.

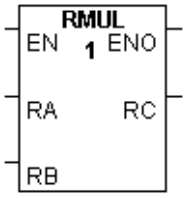


Figure 10.31 Real Multiply Function Block

Inputs

EN is the Real Multiply enable bit. A True condition of this bit causes the values of RA and RB to be multiplied and the result stored in RC. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagram program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

RA is a multiply value. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

RB is another multiply value. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

ENO is the Real Multiply enable out bit. This bit is enabled and passes power when the enable bit is True, as long as the value of RC does not go out of range. Place an output coil on the rung between Q in the function block and the right power rail in a Ladder Diagram program.

RC is the product value. RC contains the product of RA and RB as long as the enable bit is True. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, ENO is disabled. RC reverts to its last valid value whenever ENO is disabled.

Real Divide (RDIV)

The Real Divide (RDIV) function block, shown in *Figure 10.32*, produces the quotient of two real Variable values.

RDIV calculates the quotient between one value (RA) which is the dividend and another value (RB) which is the divisor and places the result in another Variable (RC) when the enable bit (EN) is True. The values of RA and RB can be preset to a constant or assigned a Variable name, which allows their value to change dynamically within the program.

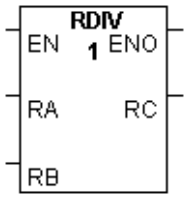


Figure 10.32 Real Divide Function Block

Inputs

EN is the Real Divide enable bit. A True condition of this bit causes the value of RA to be divided by the value of RB and the result stored in RC. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagram program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

RA is the dividend. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

RB is the divisor. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur. Also, if RB is assigned a value of zero (0), the enable out bit becomes False and RC retains its previous value.

Outputs

ENO is the Real Divide enable out bit. This bit is enabled and passes power when the enable bit is True, as long as the value of RC does not go out of range.

RC is the quotient. RC contains the quotient of RA and RB as long as the enable bit is True. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. No remainders are stored in this value; only the whole number value. If this Variable is assigned a value out of these ranges, ENO is disabled. RC reverts to its last valid value whenever ENO is disabled.

Real Move (RMOV)

The Real Move (RMOV) function block, shown in *Figure 10.33*, places the value of one Variable into another Variable.

RMOVE places the input Variable (RIN) value in the output Variable (ROUT) while the enable bit (EN) is True. The output bit (Q) always assumes the value of the enable bit; therefore, when the enable bit is True, the output bit is also True. The Variables for RIN and ROUT can be any type.

Figure 10.33 Real Move Function Block

Inputs

EN is the Real Move enable bit. A True condition of this bit causes the value of RIN to be copied to ROUT. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in a Ladder Diagrams program. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagram program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

RIN is the Real Move input Variable. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur.

Outputs

Q is the Real Move output bit. This bit is enabled and passes power when the enable bit is True. Place an output coil on the rung between Q in the function block and the right power rail (in a Ladder Diagram program) or connect an output Variable to it (in a FBD program).

ROUT is the Real Move output Variable. The value of the input Variable is copied to this Variable when the enable bit is True. The range of values this Variable can assume is from -$3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of these ranges, unpredictable results will occur. ROUT retains its last valid value whenever EN is disabled.

Integer To Real (ITOR)

The Integer To Real (ITOR) function block, shown in *Figure 10.34*, converts value of type integer to type real. The converted real value is then stored in the real Variable assigned to the output.

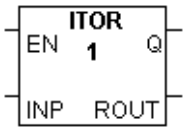


Figure 10.34 Integer to Real Function Block

Inputs

EN is the Integer To Real enable bit. A True condition of this bit causes the integer inputted to INP to be converted to a real number. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagrams program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

INP is the Integer to Real function block integer input to be converted when EN is True.

Outputs

Q is the Integer To Real function block output bit. This bit is enabled and passes power while a True condition of the enable bit occurs.

ROUT is the Integer To Real function block real-number output. This is the value of the integer passed to the IN changed to real-number format.

Real to Integer (RTOI)

The Real To Integer (RTOI) function block, shown in *Figure 10.35*, converts values of type real to type integer. The converted real value is then stored in the real Variable assigned to the output.

Figure 10.35 Real to Integer Function Block

Inputs

EN is the Real to Integer enable bit. A True condition of this bit causes the integer inputted to RIN to be converted to a real number. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN in the function block. In FBD programs, the value of EN is determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagrams program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

RIN is the Real To Integer function block real-number input to be converted when EN is True.

Outputs

Q is the Real To Integer function block output bit. This bit is enabled and passes power while a True condition of the enable bit occurs.

OUTP is the Real To Integer function block integer output. This is the value of the real-number passed to the RIN changed to integer format.

Square Root (SRT)

The Square Root (SQRT) function block, shown in *Figure 10.36 on page 156*, calculates the square root of the real value entered at its input RIN.

SQRT places the result in the Variable assigned to ROUT when the enable bit (EN) is True. The values of RIN can be preset to

a constant or assigned a Variable name, which allows their value to change dynamically within the program.

Figure 10.36 Square Root Function Block

Inputs

EN is the Square Root enable bit. A True condition of this bit causes the square root of RIN to be placed in ROUT. In a program using Ladder logic, the value of EN is determined by the Boolean evaluation of elements placed between the left power rail and EN. In FBD programs, the value of EN determined by the Variable connected to EN. If no contacts are placed on the rung before EN in a Ladder Diagram program, the value of EN is always True. In FBD, all inputs must be connected to a Variable.

RIN is the Square Root input Variable. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of this range, unpredictable results will occur.

Outputs

Q is the Square Root output bit. This bit is enabled and passes power when the enable bit is True. Place an output coil on the rung between Q in the function block and the right power rail (in a Ladder Diagram program) or connect an output Variable to it (in a FBD program).

ROUT is the Square Root output Variable. This value is set equal to the square root of the input Variable when the enable bit is True. The range of values this Variable can assume is from $-3.4x10^{38}$ to $3.4x10^{38}$. If this Variable is assigned a value out of this range, unpredictable results will occur. ROUT retains its last valid value whenever EN is disabled.

Logical

The following section describes the logical function blocks found in the *standard* category.

Logical And (AND)

The Logical AND function block, shown in *Figure 10.37*, acts exactly like a logical "and" gate. If both inputs are True, the output is enabled. On the other hand, if even one of the inputs are False, the output is disabled. All inputs and outputs of this function block must be Boolean.

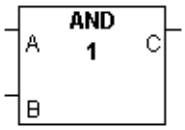


Figure 10.37 The Logical AND Function Block

Inputs

A -- Boolean input element.

B -- Boolean input element.

Outputs

C -- Boolean output element.

Logical Or (OR)

The Logical OR function block, shown in *Figure 10.38*, acts exactly like a logical "or" gate. If either or both of the inputs are True, the output is enabled. On the other hand, if all of the inputs are False, the output is disabled. All inputs and outputs of this function block must be Boolean.

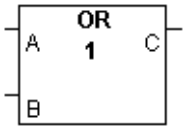


Figure 10.38 The Logical OR Function Block

Inputs

A -- Boolean input element.

B -- Boolean input element.

Outputs

C -- Boolean output element.

Vendor Provided Function Blocks

This section provides information and examples of vendor provided function blocks.

In this section, the terms **symbol 1, symbol 2,** and **symbol 3** are used. These terms refer to the labels shown within each of the function blocks as shown in *Figure 10.39*.

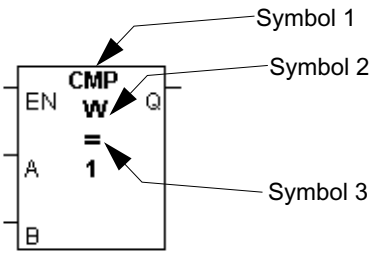


Figure 10.39 Vendor Provided Function Block diagram

Vendor provided comparison and calculation functions offer an alternate format and other options not found in standard function blocks.

Comparison (CMP)

symbol1 = CMP
The comparison function block (CMP) is designed to test quantitative relationships between two specified inputs, A and B, and report the finding in the output bit (Q). See *Figure 10.40 on page 159*.

symbol2 = W or L
The Variables A and B need to be declared as **INT** type or **LONG** integer type, of size specified in this symbol. Two memory sizes are available for these integers, word-length W (16-bit) and long-word L (32-bit). Pulse-types are not available.

symbol3 = comparison
In the CMP function block, this symbol specifies the type of comparison to be made and takes on one value from the set of { = (equal), <> (not equal), > (greater than), >= (greater or equal), < (less than), <= (less than or equal)}. For example, the function block would compute the comparison

A <= B

and enable the output bit Q, if B is greater or equal to A, and disable Q if B is less than A. Similarly, the comparison A<>B would enable Q if A does not equal B, while disable Q if A did equal B.

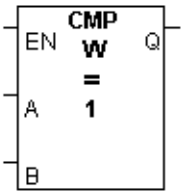


Figure 10.40 The Comparison Function Block

Inputs

EN-- is the CMP function block's enable bit. It allows the comparison to be performed if True.

A-- is the CMP function's first argument.

B-- is the CMP function's second argument.

Output

Q-- is the CMP function's output bit. The output bit is disabled if the enable bit is False, as well as if the logical operation performed computes to False.

Calculation (CALC)

symbol1 = CALC
This function block, shown in *Figure 10.41 on page 160*, implements simple arithmetic and logic computations on integer inputs A and B and reports the result in the integer output C. The calculation is performed if and only if the enable bit EN is True.

symbol2 = W, L, WP, or LP
The Variables A, B, and C need to be declared as integer type, of size specified in this symbol, as follows. Two memory models are available for these integers, word-length W (16-bit) and long-word L (32-bit). If the calculation to be performed is division (SYM3="/") or multiplication (SYM3="*"), and the second symbol is L, the outcome of the computation will be a double (64-bit) integer. In addition, the function block's calculation can be performed continuously, or only as pulses, when the enable bit (EN) switches value from false to true. In the continuous computation case, integers are marked as W or L, while in the Pulse computation case, they are marked as WP or LP.

symbol3 = comparison
In the CAL function block, this symbol specifies the exact type of calculation to be made and takes on one value as shown in *Table 10.1* .

Table 10.1 Symbol Operation

Symbol	Operation
+	addition
-	subtraction
*	multiplication
/	division
&	Logical AND
|	Logical OR
@	Exclusive XOR
@!	Exclusive NOT OR

For example, the function block would compute the operation

A * B

as (the value of A) times (the value of B) and places the result of the 16-bit integer multiplication into the destination location C. Similarly, the bitwise NOR calculation A @! B would place in destination C the result of the bitwise NOT OR (Boolean nor) comparison. If the third symbol is division (SYM3="/"), and the second argument of the calculation is B = zero, an error occurs, marked by disabling the output bit Q.

NOTE! ***If the output of an arithmetic operation (+, -, *, or) falls outside the acceptable range of C, Q is locked as disabled (0).***

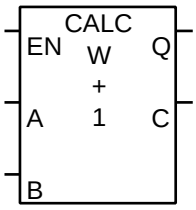


Figure 10.41 The Calculation Function Block

Inputs

EN-- is the CAL function block's enable bit. The calculation is performed if True.

A-- is the CAL function's first argument.

B-- is the CAL function's second argument.

Outputs

Q-- is the CAL function's output bit. The output bit is disabled if the enable bit is False; it is enabled as soon as the input enable bit becomes True.

C-- is the CAL function's output destination. If the calculation to be performed is addition (SYM3="+") or multiplication (SYM3="*"), and the second symbol is L, the outcome of the computation will be a double (64-bit) integer.

NOTE! ***InterLock, InterLock Clear, and Jump Function blocks are not available in FBD programs.***

Jump (JMP)

symbol1 = JMP

The Jump (JMP) function block, shown in *Figure 10.42*, performs a GOTO <LABEL> action. This function block needs to be positioned at the end of a rung. Labels can be set at rungs throughout the Ladder Diagram by clicking on the rung number in the workspace. Infinite loops cause errors which can be detected by scan-time monitoring.

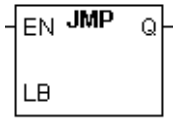


Figure 10.42 The Jump Function Block

Inputs

EN-- is the Jump enable bit.

LB-- is the Label of the rung to which the function block points and passes control to in a GOTO-like fashion.

See *"Assigning Labels to Rungs" on page 100* for information on labeling rungs.

Output

Q-- is the output Jump function block's output bit. Its value is always equal to that of the enable bit EN.

InterLock (IL)

InterLock and InterLock-Clear function blocks, shown in *Figure 10.43* and *Figure 10.44* are designed to act as opening (IL) and closing (ILC) brackets to a segment of logic. The section(s) of a ladder program positioned between IL and ILC rungs can be selectively executed. IL and ILC cannot be nested. IL and ILC alter the performance of all elements contained in the "bracket" (all rungs between the IL and the ILC markers) as dictated by the result of the logic to the left of the IL block. Therefore, if the EN bit of the IL block is False, there is no execution of the bracketed logic; however, if the EN bit of the IL block is True, the bracketed logic executes. The IL/ILC bracket is local to one Ladder, and so does not affect others which may be running simultaneously in a "parent" SFC.

symbol1=IL
The InterLock function block marks the beginning of a group of Ladder logic diagram operations that are "bracketed" for special purpose.

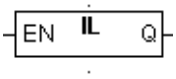


Figure 10.43 The InterLock Function Block

Input

EN-- is the InterLock enable bit. A True condition of this bit activates the IL/ILC bracket and causes the special effects listed in the previous section.

Output

Q-- is the InterLock output bit. This bit reflects the state of the enable bit (EN). It does not affect the rest of the program.

InterLock-Clear (ILC)

symbol1=ILC
The ILC function block simply marks the end of the Ladder logic segment that is affected by the Interlock function.

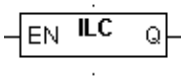


Figure 10.44 The InterLock Clear Function Block

Input

EN-- is the InterLock-Clear enable bit. In this Function Block, it has no effect on the program. The ILC FB closes the IL bracket whenever the preceding IL FB is active.

Output

Q-- is the InterLock-Clear output bit. This bit is always enabled. It does not affect the rest of the program since the ILC function block's position is always at the end of a rung.

Rotation (RROT/LROT)

symbol1 = RROT, LROT

The rotation function, shown in *Figure 10.45*, performs a bit-level rotation of 16-bit or 32-bit integers: the right rotation RROT moves the i-th bit into the (i-1)'th bit, as the lowest bit is carried into the highest; the left rotation LROT moves the i-th bit into the (i+1)'th bit, as the highest bit is carried into the lowest bit. The operation is performed on either BIN or BCD numerals which are treated simply as a collection of bits, such that range-checking is not performed.

symbol2 = W, L, WP, or **LP**

The Variable D needs to be declared as integer type, of size specified in this symbol, as follows. Two memory sizes are available for this integer, word-length W (16-bit) and long-word L (32-bit). In addition, the function block's calculation can be performed continuously, or only as pulses, when the enable bit (EN) switches value from False to True. In the continuous computation case, integers are marked as W or L, while in the Pulse computation case, they are marked as WP or LP.

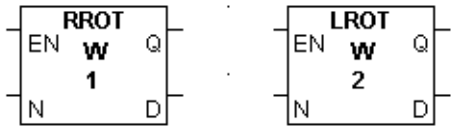


Figure 10.45 The Right and Left Rotation Function Blocks

Inputs

EN-- is the rotation function block's enable bit: the conversion is performed if it is True.

N-- is an integer (**INT**) or long integer (**LONG**) specifying the number of bits the rotation should be done for per scan.

D-- is the **INT** or **LONG** integer Variable on which the rotation is to be performed (also an output).

Outputs

Q-- is the rotation function's output bit. The output bit is disabled if the enable bit is False and is enabled as soon as the enable bit is True.

D-- is the rotation function's output destination (as well as input). The input **INT** or **LONG** integer will be replaced with the result of the rotation operation.

Shift (RSFT/LSFT)

symbol1 = RSFT, LSFT
The shift function, shown in *Figure 10.46*, performs a bit-level shift of 16-bit or 32-bit integers: the right shift RROT moves the i-th bit into the (i-1)'th bit, as the lower bits are lost and the higher padded with zeros; the left rotation LROT moves the i-th bit into the (i+1)'th bit, as the highest bits are lost and the lower bits padded with zeros. The operation is performed on either BIN or BCD numerals which are treated simply as a collection of bits, such that range-checking is not performed.

symbol2 = W, L, WP, or **LP**
The Variable D needs to be declared as integer type, of size specified in this symbol, as follows. Two memory sizes are available for this integer, word-length W (16-bit) and long-word L (32-bit). In addition, the function block's calculation can be performed continuously, or only as pulses, when the enable bit (EN) switches value from False to True. In the continuous computation case, integers are marked as W or L, while in the Pulse computation case, they are marked as WP or LP.

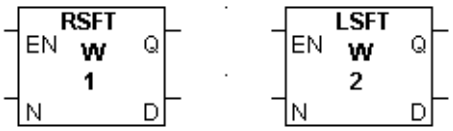


Figure 10.46 The Right and Left Shift Function Blocks

Inputs

EN-- is the shift function block's enable bit: the conversion is performed if it is True.

N-- is an integer (**INT**) or long integer (**LONG**) specifying the number of bits the shift should be done for per scan.

D-- is the **INT** or **LONG** integer Variable on which the shift is to be performed (also an output).

Outputs

Q-- is the shift function's output bit. The output bit is disabled if the enable bit is False and is enabled as soon as the enable bit is True.

D-- is the shift function's output destination (as well as input). The **INT** integer or **LONG** integer will be replaced with the result of the shift operation.

Negation (NEG)

symbol1 = NEG
The NEG function block, shown in *Figure 10.47*, implements a bitwise-NOT and complements the lowest bit (two's position).

symbol2 = W, L, WP, or **LP**
Variable A and B need to be declared as integer type, of size specified in this symbol, as follows. Two memory sizes are available for this integer, word-length W (16-bit) and long-word L (32-bit). In addition, the function block's calculation can be performed continuously, or only as pulses, when the enable bit (EN) switches value from False to True. In the continuous computation case, integers are marked as W or L, while in the Pulse computation case, they are marked as WP or LP.

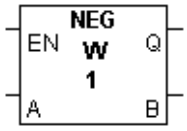


Figure 10.47 The Negation Function Block

Inputs

EN-- is the NEG function block's enable bit: the conversion is performed if it is True.

A-- is the integer (**INT**) or long integer (**LONG**) Variable to have NEG performed on it.

Outputs

Q-- is the NEG function's output bit. The output bit is disabled if the enable bit is False and is enabled as soon as the enable bit is True.

B-- is the NEG function's output destination (as well as input). The **INT** integer or **LONG** integer will be replaced with the result of the NEG operation.

11

User Defined Function Blocks

LogicPro provides comprehensive functionality to create, edit and use User Defined Function Blocks (UDFB). Users can write their own Function Blocks in C and use them within a LogicPro Ladder Program.

Overview

UDFBs are very powerful tools you can use to handle such tasks as:

- Support a specific hardware device.
- Performing an application specific control function.
- Performing complex math equations.

UDFBs can greatly simplify your Programs and facilitate troubleshooting.

Getting Started

You can use User Defined Functions Blocks in the same manner as standard Function Blocks once:

- UDFBs are defined.
- Blocks are coded.
- Blocks are added to the library.

Once a UDFB is defined in the resource, it is valid and usable in all Programs.

When a UDFB is defined, *LogicPro* generates two associated files:

- A header file.
- A C code file.

The header file contains the definition of a structure for that UDFB and function prototypes. The C file contains four functions:

Main Program - called during evaluation of the user Program and performs the task of the UDFB.

Initialization - called only once, when the user Program starts to run.

Reset - called only when the UDFB is in an unstored action and that action becomes inactive.

Exit - called once, if the user Program terminates.

In most cases, you will not use the initialization, exit, and reset functions; therefore, you will not need to add code to those functions.

Defining a New or Editing an Existing UDFB

In order to create a new UDFB the user must first open or create a Project and a Resource.

To create a new UDFB:

1. Select the **File** menu.
2. Select **New.**
3. Select **User Defined Function Block**. The **New UDFB** dialog box appears, as shown in *Figure 11.1 on page 169*.
4. Enter the complete name of the UDFB in the **Long** field. This is the name of the UDFB as it appears in the list of available UDFBs in the Ladder diagram editor. The Long name is limited to 27 characters or less.
5. Enter an abbreviated name for the UDFB in the **Short** field. The Short name is the label that is placed at the top of the graphical representation of the Function Block as it appears in the Program editor.

   The Short name must:
   - Use a letter from A to Z for the first character.
   - Be 4 characters or less in length.
   - Not use a space or other illegal character.

6. Define the Variables that will be used as inputs and outputs, as well as internally by the Function Block. Refer to *"In's, Out's and Internals"* below.

7. Click **[OK]** to create the new UDFB
– or –
[Cancel] to terminate the process.

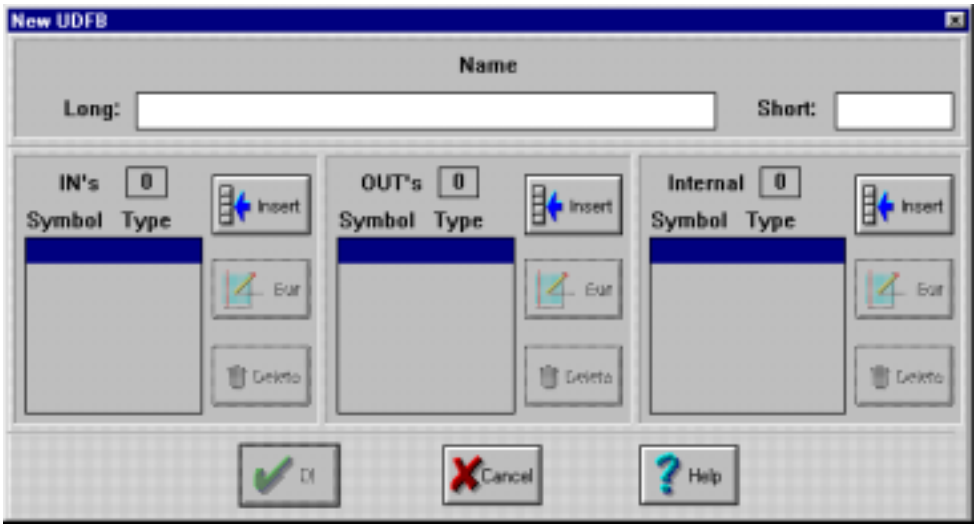


Figure 11.1 New UDFB Dialog Box

In's, Out's and Internals

IN's - contains the symbol and type of Inputs to the UDFB.

OUT's - contains the symbol and type of Outputs from the UDFB.

Internal - contains the symbol and type of internal assignment. They may also be used as extra inputs and outputs, but they will not be displayed. The major differences between Internals and Ins are that internals are:

- Not displayable in Ladder diagrams.
- Only constants.

NOTE! ***Variables can be used, but not changed on-line.***

The *symbol* is the name of the particular input, output or internal when the UDFB is graphically represented in the Ladder diagram, and must be unique within the UDFB. LogicPro does not allow two symbols of the same name in one UDFB.

The symbol must:

- Use a letter from A to Z for the first character.
- Be four characters or less in length.
- Not use a space or other illegal character.

The *type* defines what kind of variable you can assign to the particular input.

A UDFB may have up to five inputs or outputs, but the first input and output must be Boolean, and all Boolean inputs must come before any other type of inputs. There must be at least one Boolean input and at least one Boolean output. The input controls enable the block outputs. The output should show that the block is active.

A UDFB may have up to fifteen internals. Internals are an option for a UDFB. They aren't required.

Operations

There are three operations associated with each **IN's, OUT's** and **Internal** group:

- Insert
- Edit
- Delete

NOTE! ***You can not Edit or Delete if no symbols are defined.***

To insert an Input, Output, or Internal:

1. Select the position in which to insert your symbol. If there is a symbol defined in that position already, that symbol will be moved down one spot and the new symbol will be placed in that position.
2. Click **[Insert].** The **Add Symbol Definition** dialog box, shown in *Figure 11.2 on page 171*, appears.
3. Enter the name of the symbol in the **Name** field.
4. Select the type of symbol from the **Type** field.
5. Click **[OK]** to add the symbol to the group.

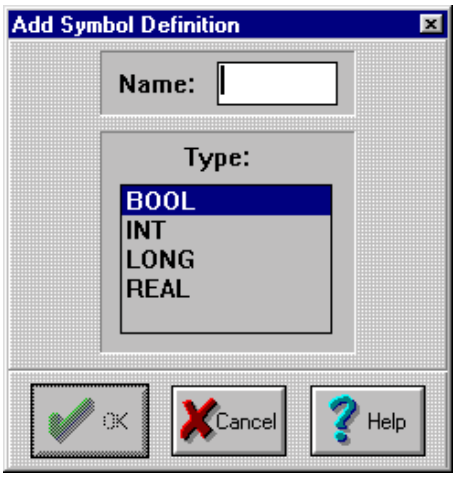


Figure 11.2 Add Symbol Definition Dialog Box

To edit an Input, Output, or Internal:

1. Select a symbol by clicking on it.
2. Click **[Edit]**. The **Edit Symbol Definition** dialog box, shown in *Figure 11.3*, appears.
3. Edit the name of the symbol in the **Name** field, if necessary.
4. Edit the type of symbol from the **Type** field, if necessary.
5. Click **[OK]** to replace the old symbol with the new symbol in the group.

NOTE! ***When type is "Real" the name MUST start with the letter 'R.'***

Figure 11.3 Edit Symbol Definition Dialog Box

UDFB Files

As soon as a new UDFB is defined, LogicPro automatically generates two files for the UDFB, one header file and one C file. The header and C file have the short name of the UDFB with a prefix indicating in which resource it has been defined. For example, PPCCNT.h and PPCCNT.c are the names for the header and C files if the UDFB has a short name of CNT and is defined for PPC-2000.

The header file contains the definition of a structure for that UDFB and function prototypes. The structure is similar to the following example:

```
typedef struct {

/*Input and output symbols */

}XXXXStruct;
```

The XXXX will be replaced by the short name of the UDFB. The members of the structure are the inputs, outputs and internals. The user should not change these. However, the user may add additional members into the structure as needed. Note that Boolean variables are included in this structure. These variables are used by Function Block instances.

The C file contains four functions which may look like the following example:

```
int XXXX(bool EN,bool *Q,int *C,XXXXStruct *XXXXStructA);

void InitXXXX(XXXXStruct *XXXXStructA,int A,int B);

void ResetXXXX(XXXXStruct *XXXXStructA);

void ExitXXXX(XXXXStruct *XXXXStructA);
```

The XXXX will again be replaced by the short name of the UDFB.

Main Function

The main function, shown in the following example, is called during evaluation of the user Program for every scan and performs the task of the UDFB.

```
int XXXX(bool EN,bool *Q, all other variables defined,*XXXXStruc-
tA);
{
return 0
}
```

All Boolean inputs and outputs are passed as parameters. However, Boolean inputs are passed as copies of the variable while Boolean outputs are passed as pointers. This is because the outputs use a pointer to exchange the value from the Function Block to the variable specified by the Ladder diagram. For example, if the UDFB wants to turn on Q, it must reference Q as a pointer. An instance of structure pointer is also passed to the main function, this structure points to the variable data.

Initialization Function

The initialization function, shown in the following example, is called once when the user Program starts to run.

```
void InitXXXX(XXXXStruct *XXXXStructA,intig A,intig B);
{
WALK StructA->A=A
:}
```

The instance of the structure pointer and non Boolean inputs variable are passed to the function. The initialization function provides the user the opportunity to initialize the structure members or initialize certain hardware devices before the main function is called in the user Program for every scan.

Reset Function

The reset function, shown in the following example, is only called if the UDFB is in an unstored action and that action becomes inactive.

```
void ResetXXXX(XXXXStruct *XXXXStructA);
```

When an unstored action becomes inactive, it resets all variables to the original state that they where in upon entering the action. This function should be used to reset internal variables or flags that may have changed during the course of the action being active. In many cases, the function can be empty.

Exit Function

The exit function, shown in the following example, is called if the user Program terminates gracefully.

```
void ExitXXXX(XXXXStruct *XXXXStructA);
```

In many cases, the function can be empty. For others, the function may reset a hardware device, such as a communication port.

Compiling and Building a Library

Once the code is written and is ready to test, the user can either:

- Compile the UDFB file.
- Build a library.

To compile the UDFB file from within the editor window:

1. Select the **File** menu.
2. Select the **UDFB** menu.
3. Select **Compile for Execution**
   -or-
   Compile for Simulation.

Either of these options causes *LogicPro* to compile your UDFB. A DOS screen will appear momentarily as the compiler works. Once the DOS screen disappears, your UDFB is compiled. The simulation library will be used in simulation mode and the execution library in runtime.

To build a library:

1. Select **Build Execution Lib** from the **UDFB** menu. The **Build UDFB Library for Execution** dialog box, shown in *Figure 11.4*, appears.
2. Click the UDFBs that you want to put into the library since only the selected UDFBs will be in the finished library.
3. Click **[OK]** to compile the files and build a library.

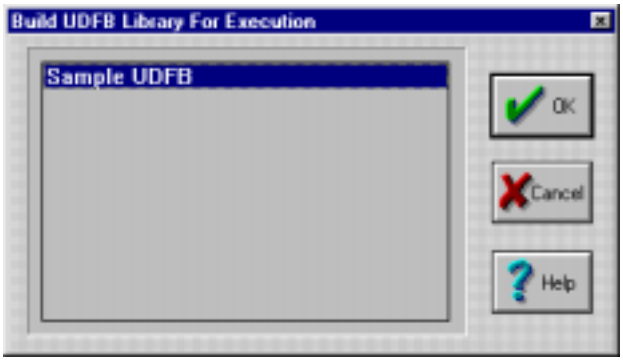


Figure 11.4 Build UDFB Library for Execution

The library is named `xxxUDFB.lib` with a prefix indicating the kind of resource that is currently opened.

The library is stored in the following directory:

```
\logicpro\user\bc
```

After the UDFB library is successfully built up, you can use the UDFB in a Ladder diagram or Function Block Diagram Program as a standard Function Block. When your Program is compiled and linked for execution, the library is automatically linked by *LogicPro*. All the functionality in *LogicPro* which is available to standard Function Blocks is also available to User Defined Function Blocks.

UDFB in Simulation

UDFBs are most often used to perform some application specific tasks. Some of them may be hardware specific. However, simulation in *LogicPro* can run on a PC that may not be able to perform such hardware specific tasks. The user may want to simulate hardware which is not available in simulation mode.

In order to overcome the problem, a pre-defined macro, `SIMULATION`, is automatically inserted if **Compile for Simulation** or **Build Simulation Lib** is selected from the UDFB menu. You must use:

- `#ifndef SIMULATION`
- `#endif`

to exclude all hardware and target operation system specific code for conditional compiling. The pre-defined macro, `SIMULATION`, also allows you to write code in order to make the UDFB behave the way you want it to in simulation.

To have a UDFB run in both simulation and the execution environment, refer to the following example:

```
int XXXX(bool EN,bool *Q,int *C,XXXXStruct *XXXXStructA)

{

#ifndef  SIMULATION

    all the hardware and target operation specific code

#endif

#ifdef  SIMULATION

    code that user wants in simulation

#endif

}
```

To build a UDFB in simulation mode:

1. Select the **File** menu.

1. Select the **UDFB** menu
2. Select **Build Simulation Lib**. The **Build UDFB Library for Simulation** dialog box appears, as shown in *Figure 11.5*.

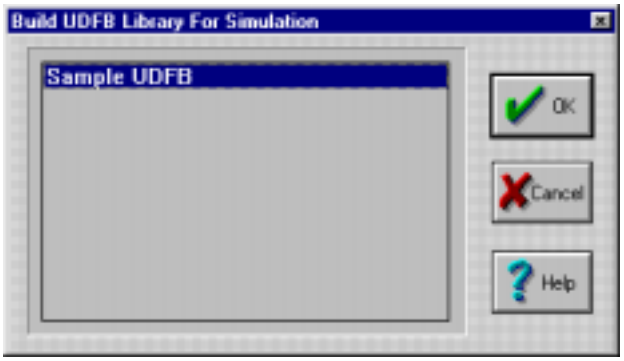


Figure 11.5 Build UDFB Library for Simulation Dialog Box

3. Select the UDFBs that you want to include in the library. Only those UDFBs which are selected will be in the library.
4. Click **[OK]**. *LogicPro* compiles these files and builds a library with the pre-defined macro `SIMULATION`.

The library is named `UDFB.lib` with a prefix indicating the kind of resource that is currently open.

The library is stored in the following directory:

```
\logicpro\simulate\bc
```

After the simulation UDFB library is built successfully, you can use the UDFB in a Ladder diagram as a standard Function Block in simulation. When the Program is compiled and linked for simulation, the simulation library is automatically linked by LogicPro.

Working with UDFB Files

The following sections describe how to work with UDFB files.

Editing UDFB Files

When UDFB header or C files are active on the window, the Undo, Cut, Copy and Paste commands in the Edit menu are enabled. You can use these functions to edit text.

Opening UDFB Files

To Open UDFB files:

1. Select **File**.
2. Select **Open**.
3. Select **User Defined Function Block** to open the header and C files of the selected UDFB. The **Open User Defined Function Block** dialog box, shown in *Figure 11.6*, appears.
4. Select the UDFB you want to open.
5. Click **[OK]**.

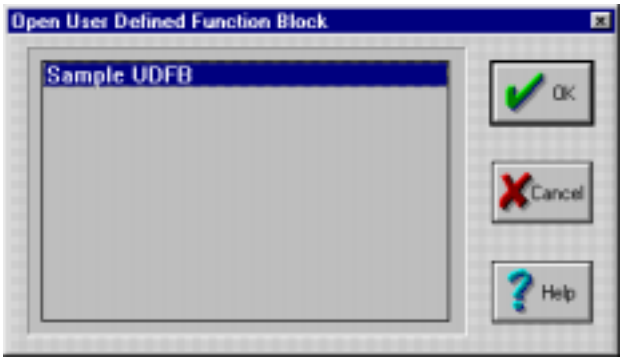


Figure 11.6 Open User Defined Function Block Dialog Box

Closing UDFB Files

To Close UDFB files:

1. Select **File**.
2. Select **Close**.

3. Select **User Defined Function Block** to close the header and C files of the selected UDFB.
4. Select the UDFB you want to close. You will be prompted to save any files that have changed.
5. Click **[OK]**.

Saving UDFB Files

You can save the UDFB header and C files by selecting **Save** from the **File** menu.

Deleting UDFBs

To delete UDFBs:

1. Select **File**.
2. Select **Delete**.
3. Select **User Defined Function Block** to remove the files for one or more UDFBs. The **Delete User Defined Function Block** dialog box appears, as shown in *Figure 11.7*.
4. Select the UDFBs you want to delete.
5. Click **[OK]** to delete the UDFBs
   –or–
   [Cancel] to abort the action.

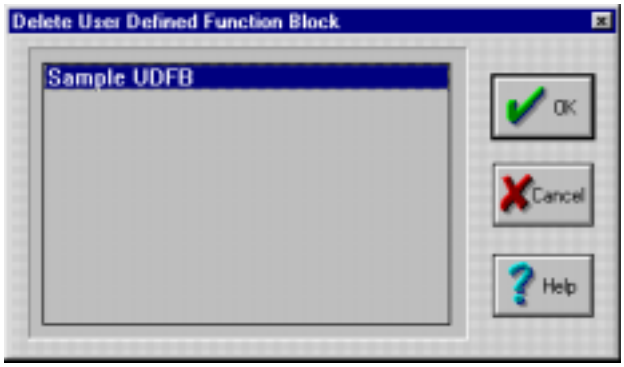


Figure 11.7 Delete User Defined Function Block Dialog Box

UDFB Tutorial

The following section is an example for writing a very simple User Defined Function Block that adds two numbers. This section will give a detailed explanation for each step as the development of the UDFB continues.

Selecting a Resource

The first step in writing a UDFB is to open a resource. UDFBs are tied to specific resource platforms because of hardware and target operation system specific dependencies.

UDFB Definition

The next step is to define the UDFB and how it will relate to the *LogicPro* application Program. In this example, the UDFB is expected to add two integers and assign the sum to a third integer. You may also want to control when the addition occurs through the Ladder logic.

Once the functionality of the UDFB is certain, the next step is to design the interface between the UDFB and the *LogicPro* application. This is done by listing the inputs and outputs.

The first Boolean input and output should be:

- An enable input to enable the addition.
- An output that indicates that the addition is valid and that the UDFB is enabled.

These input and output are Boolean because they have either a true or false state.

Three operands are needed to perform the addition. Operand C will equal Operand A plus Operand B if the enable is true. These inputs and output are of integer type. The range of an integer is -32,768 to 32,767.

Finally the two names for the Function Blocks must be defined. One name is a long name that appears in the list of UDFBs when programming Ladder logic. This name can be up to 27 characters in length. The other is the short label that appears at the top of the UDFB in the graphical Ladder environment. This label can be up to 4 characters in length.

Figure 11.8 shows how the Function Block looks in a program.

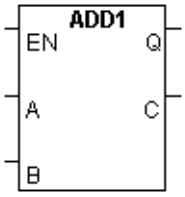


Figure 11.8 ADD1 Function Block

Use the following steps to define the UDFB:

1. Select **File**.
2. Select **New**.
3. Select **User Defined Function Block**. The **Add UDFB** dialog box appears.
4. Enter `Example 1 Add` as the Long Name in the **Long** field.
5. Enter `ADD1` as the Short Name in the **Short** field.
6. Click **[Insert]** in the **Inputs** group.
7. Name the first input `EN` and set its **Type** to **Bool**.
8. Click **[OK]**.
9. Repeat steps 6 to 8 with the second and third inputs Named as `A` and `B` with **Type** set to **INT**.
10. Click **[Insert]** in the **Outputs** group.
11. Name the first output as `Q` and set its **Type** to **Bool**.
12. Click **[OK]**.
13. Repeat steps 10 to 11 with the second output named as `C` with **Type** set to **INT**.
14. Click **[OK]**.

LogicPro automatically creates two files, a header file (`*.h`) and a C file (`*.c`), and then puts these two text files on the window. Some of the names will differ if you are using a CPC400. In the header file, there is a structure named `Add1Struct` defined along with four function prototypes. In the following example, it is not necessary to add additional members to the structure.

In the C file, there are four functions. It isn't necessary to write any additional code in the **Init, Reset** and **Exit**

functions. The structure and these three functions are listed in the following example as they are generated by *LogicPro*.

```
#ifndef PCADD1UDFBHeader
#define PCADD1UDFBHeader

#define bool char

#define Int16 short

typedef struct {
    Int16 A;
    Int16 B;
    Int16 C;
    bool EN;
    bool Q;

}ADD1Struct;

extern int ADD1(bool EN,bool *Q,ADD1Struct *ADD1StructA);
extern void InitADD1(ADD1Struct *ADD1StructA,Int16 A,Int16 B);
extern void ResetADD1(ADD1Struct *ADD1StructA);
extern void ExitADD1(ADD1Struct *ADD1StructA);
#endif
```

Let's take a closer look at the main function and how to write C code.

```
int ADD1(bool EN,bool *Q, ADD1Struct *ADD1StructA);
```

This function is called in the evaluation loop of your Program. It is called when the rung of Ladder logic that it resides on, or a Function Block diagram, is being evaluated.

All Boolean inputs are given to the function passed by their copies. All other inputs are passed by reference through the structure. In our example, ADD1, use the structure pointer ADD1 StructA to get the value of either A or B.

Any output can use a pointer to exchange the value from the Function Block to the variable specified by the Ladder diagram. In this example, we use a pointer C to pass the value of our arithmetic operation to the variable assigned in the Ladder diagram.

The final parameter that is passed to the function is a pointer to a structure.

```
typedef struct {
    int A;
    int B;
    int C;
    bool EN;
    bool Q;
}ADD1Struct;
```

This structure holds all variables that are passed to the Function Block. New members can be added to the end of this structure. These new members will then retain their value even after the function exits. Each instance of the UDFB will have its own separate instance of the structure, so interaction between several instances of the same Function Block can be prevented.

Boolean variables are created within this structure to match the boolean variables defined for the block. These variables are used for Function Block instances. The user has the option of updating these variables. As a general rule they should be set to match the Function Block variables. ADD1 C code can be written as in the following example:

```
#include “PCADD1.h”

int ADD1(bool EN,bool *Q,ADD1Struct *ADD1StructA)
{
long Result;
    /* Declare long variable to hold result */
    /* even if addition result is out of integer range */

/* Check that block is enabled */
if (EN)
    {
    /* Perform addition */
    Result = (long) ADD1StructA->A + ADD1StructA->B;

    /* Check that result is within int range (-32768 to 32767) */
    if ((Result >= -32768) && (Result <= 32767))
    {

    /* Set Bool output to signify successful completion */
    *Q = 1;

    /* Assign Result to integer output */
    ADD1StructA->C = Result;

    /* Copy bools to structure for access as Function Block
    instance variables */
    ADD1StructA->EN = EN;
```

```
    ADD1StructA->Q = *Q;
    }
else
    {
    *Q = 0;
    }
  }
return 0;
}
void InitADD1(ADD1Struct *ADD1StructA,Int16 A,Int16 B)
{
  ADD1StructA->A = A;
  ADD1StructA->B = B;
}

void ResetADD1(ADD1Struct *ADD1StructA)
{
}

void ExitADD1(ADD1Struct *ADD1StructA)
{
}
```

12

Input/Output Drivers

Logic program variables may be internal or they may hold and set values associated with physical I/O, closed-loop control parameters or other memory registers. Internal variables are not available to the closed-loop control program, AnaWin, an OIT or any other Human Machine Interface (HMI) software. Internal variables are only available to the logic program and the LogicPro programming and monitoring package. All variables that are not internal to the logic program are accessed using I/O drivers.

Several drivers have been provided for the controllers. All the drivers allow logic program variables to be linked to registers in the database. The database stores all the control parameters such as setpoints, process variables, PID constants, alarm limits and so on. The database also contains the I/O image for all the digital and analog I/O. There is also space reserved in the database that can be accessed both by a logic program and an HMI program. By linking a variable to the database, a logic program can, for example, change the state of an output, read that state of an input, or change a closed-loop control setpoint for a channel.

To create a variable that uses an IO Driver, you will make a selection for each of the following:

- Variable Type
- Input or Output
- IO Size
- IO Driver

Finally you will enter the IO Physical Address that specifies the piece of information with which you want to link the Variable. The following sections describe these choices.

NOTE! ***If an output Variable in a logic Program sets a value using an IO Driver and IO Physical Address, the associated register in the controller cannot be written to by the closed-loop control program, an HMI, or any other device.***

Variable Types

Variables in a logic program can be declared to be one of four types. *Table 12.1* describes the options for Variable Type.

Table 12.1 Variable Types

Type	Description
BOOL	Boolean variables can hold values of 0 or 1 only (1 bit). This is the type of variable to create to associate with coils in a ladder diagram or transitions in an SFC. Choose BOOL for variables that will be associated with digital I/O. A BOOL variable always has its IO Size set to BIT.
INT	Integer variables can hold values from -32,768 to 32,767 (16 bits). Use this variable type with analog I/O and most closed-loop control parameters such as setpoint. Always choose WORD for the IO Size of an INT variable
LONG	Long integer variables can hold values from -2,147,483,648 to -2,147,483,647 (32 bits). LONG variables are always internal. You cannot associate a LONG variable with an I/O point or closed-loop control parameter.
REAL	Real variables (also known as floating point) hold positive and negative real numbers between -3.4e38 and 3.4e38. REAL variables are always internal. You cannot associate a REAL variable with an I/O point or closed-loop control parameter.

NOTE! ***The IO Driver and IO Size lists are sensitive to the choice you make in the Type list.***

Input/Output

In order to associate a variable with an I/O point or a closed-loop control parameter, you must select either Input or Output in the **Add Program/Resource Variable** dialog box. *Table 12.2* describes these options.

Table 12.2 Input and Output

Option	Description
Input	The value of the I/O point or closed-loop control parameter is read but not changed by the logic program. **Note:** The value of an output, set by the closed-loop control program may be an input to a logic program. You could, for instance, want to know when a control output exceeds 90% duty cycle. In that case you would set up a variable as an input that reads the percent output power from a channel.
Output	The value of the digital or analog output or closed-loop control parameter is set by the logic program. **Note:** A digital or analog input to the controller would never be associated with an output variable in a logic program. The logic program output variable can, however, set the value of a Soft Input register that in turn is treated as an analog input to a closed-loop control channel.

IO Size

The controller IO Drivers support variables defined with a size of either BIT or WORD. *Table 12.3* describes the IO Size options used with IO Drivers.

Table 12.3 IO Sizes Used with IO Drivers

IO Size	Description
BIT	Select BIT as the IO Size for a variable that holds the value of a digital I/O state. Certain other parameters in the controller are of BIT size.
WORD	Select WORD as the IO Size for a variable that holds an analog I/O value. Most parameters in the PPC-2000 are of WORD size. Word sized variables are used for analog I/O and other database parameters such as setpoint and alarm limits.

PPC-2000 IO Drivers

IO Driver Choice

The IO Driver allows the logic program to examine or set an I/O value or other value stored in the database. Choose the appropriate driver for the value which you want to access. *Table 12.4* describes what data can be accessed with each IO Driver.

Table 12.4 Data Accessed by the IO Drivers

IO Driver	Data Accessed
Processor_2010	Digital input and output states and count and frequency input values on a PPC-2010 Processor module.
Analog_202x	Analog input values on any PPC-202x series Analog Input module.
Encoder_Analog_2030	Count and frequency input values and analog output values on a PPC-2030 Encoder In Analog Out module.
Digital_IO_2040	Digital input and output states and count and frequency input values on a PPC-2040 Digital I/O module.
Analog_Out_205x	Analog output values on any PPC-205x Analog Out module.
Digital_Out_206x	Digital output states on any PPC-206x Digital Out module.
Digital_In_207x	Digital input states on any PPC-207x Digital In module.
Setpoint	The setpoint of any of the closed-loop control channels.
Soft_Input	The value of any of the Soft Inputs. Typically used for a logic program to output a value to the input of a closed-loop control channel.
Soft_Bool	The value of any Soft Bool register. Typically used to exchange information between a logic program and an HMI program or operator interface panel.
Soft_Int	The value of any Soft Int register. Typically used to exchange information between a logic program and an HMI program or operator interface panel.
Database	The value of any closed-loop control parameter that is not supported by another IO Driver.

IO Physical Addresses

The IO Physical Address syntax depends on the IO Driver. The Custom Interfacing chapter of the PPC-2000 User's

Guide lists the addresses for each I/O point. AnaWin includes the IO Physical Address in the names that appear in the SpreadSheet Overview screen. In general there are three types of I/O drivers. *Table 12.5* lists the IO Drivers in the categories that characterize the IO Physical Addresses used with them.

Table 12.5 Categories of IO Drivers

Category	Drivers
Module Specific	Processor 2010 Analog_In_202x Encoder_Analog_2030 Digital_IO_2040 Analog_Out_205x Digital_Out_206x Digital_In_207x
Parameter Specific	Setpoint Soft_Input Soft_Bool Soft_Int
Database	Database

The following sections describe the address syntax used with each category of IO Driver.

Module Specific Drivers

The module specific IO Drivers (such as Processor_2010 and Analog_In_202x) are used to access digital I/O states and analog I/O values. The specific I/O value addressed uses either two-place or three-place addresses depending on how many types of I/O the corresponding module supports. Modules that have only one type use two-place addresses. Modules that have more than one type use three-place addresses. See *Table 12.6* for a description of these syntaxes.

Table 12.6 Address Syntax for Module Specific Drivers

Syntax	Used with these IO Drivers
Two-Place Address: X.Z X = Module Address Z = Module I/O Number	Modules with only one type of I/O: Analog_In_202x Analog_Out_205x Digital_Out_206x Digital_In_207x
Three-Place Address: X.Y.Z X = Module Address Y = I/O Type Z = Module I/O Number	Modules with more than one type of I/O: Processor_2010 Encoder_Analog_2030 Digital_IO_2040

The module address is determined by the rotary switch setting on the face of the module with the exception of the processor module which has address zero.

The module I/O number is generally the number on the terminal to which the I/O device is attached, but see the PPC-2000 User's Guide to be certain.

The I/O type in the address is a digit that indicates the type of I/O addressed. Used when addressing I/O in modules that handle more than one type. *Table 12.7* lists the options for the I/O Type digit in the three-place address.

Table 12.7 I/O Type Digit in IO Physical Address

Y	I/O Type
0	Digital I/O State
1	Count Value
2	Frequency Value
3	Analog Output Value

Parameter Specific Drivers

The parameter specific IO Drivers (Setpoint, Soft_Input, Soft_Bool, and Soft_Int) are used to read or set values for specific parameters. The specific parameter value is addressed with its database offset.

The database offset is the location of a particular value relative to the first value of that parameter. For example, to access channel 17's setpoint with the Setpoint IO Driver, the IO Physical address will consist of the database offset, 17. *Table 12.8* lists the parameter specific drivers and the corresponding database offset ranges.

Table 12.8 Addresses Used with Parameter Specific IO Drives

IO Driver	Database Offset
Setpoint	Channel Number (1-48)
Soft_Input	Soft Input Number (1-50)
Soft_Bool	Soft Bool Number (1-1000)
Soft_Int	Soft Int Number (1-100)

Database Driver

The Database IO Driver is used to examine or set values for any closed-loop control parameter such as Control Mode that is not supported by one of the other drivers. The Database driver allows access to all controller parameters. Other drivers are provided for faster access to specific parameters. Of course, controller parameters that are read-only cannot be adjusted. The specific parameter value is addressed with the

parameter number and its database offset. See *Table 12.9* for a description of this syntax.

Table 12.9 Address Syntax for Database Driver

Syntax	Used with this IO Driver
#.D # = Parameter Number D = Database Offset	Database

The parameter number is a numeric identifier used to address a closed-loop control parameter. Each parameter in the PPC-2000 has a parameter number. See the Custom Interfacing chapter of the PPC-2000 User's Guide for a list of parameter numbers and descriptions.

The database offset is the location of a particular value relative to the first value of that parameter. For example, each channel has a Heat Proportional Band setting. If you want to set the Heat Proportion Band for channel 9, the address will include the database offset 9.

Four types of parameters are accessed using the Database I/O driver: Analog and Encoder Input, Channel, Digital I/O and Global. The following sections describe how to determine the database offsets to be used to access each of these types of parameters.

Channel Parameters

The database offsets associated with channel parameters are the channel numbers. Refer to the PPC-2000 User's Guide for a complete description of channel parameters and a list of channel parameter numbers.

Analog, Count, and Frequency Input Parameters

The database offsets associated with analog, count and frequency inputs are based on the order of the inputs in the database. Refer to the Custom Interfacing chapter of the PPC-2000 User's Guide for a complete description of input parameters and a list of input parameter numbers.

Digital I/O Parameters

The database offsets associated with digital inputs and outputs are based on the order of the digital I/O in the database. Refer to the Custom Interfacing chapter of the PPC-2000 User's Guide for a complete description of digital I/O parameters and a list of digital I/O parameter numbers.

Global Parameters

Global parameter offsets are dependent on the global parameter. Refer to the Custom Interfacing chapter of the PPC-2000 User's Guide for a complete list of global parameter numbers and their corresponding offsets.

Using the IO Drivers

The following sections describe and illustrate the use of each IO Driver.

Processor_2010

The Processor_2010 driver supports the digital I/O on board the PPC-2010 Processor module.

To associate a variable with a digital I/O point:

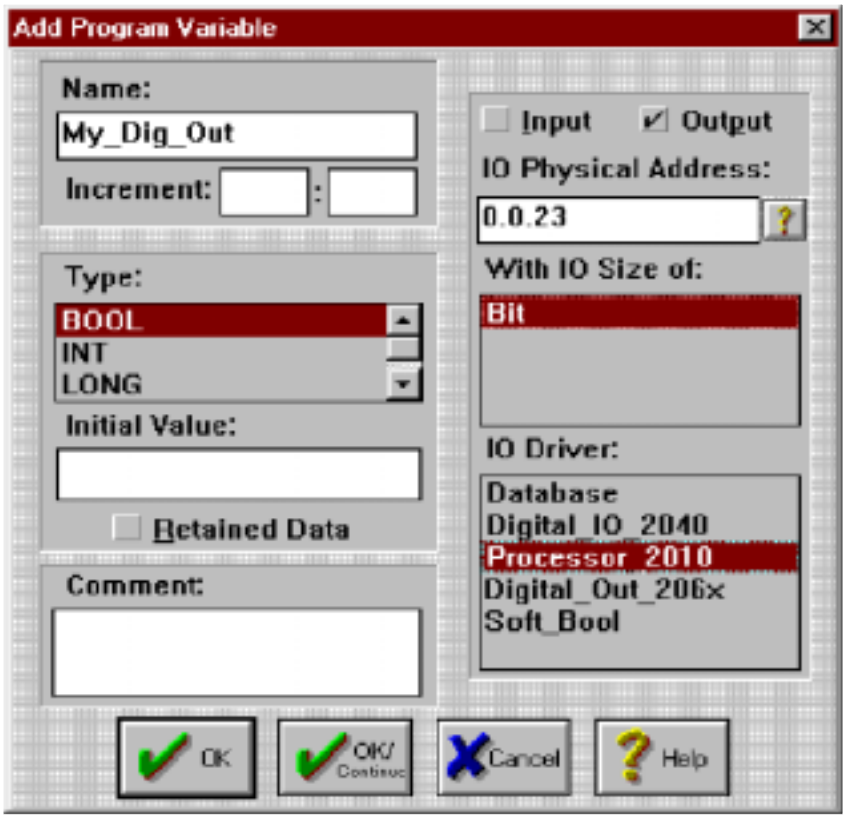
1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **BOOL** from the Type list.
4. Select **Input** or **Output** by clicking the corresponding check box.
5. Select **Processor_2010** from the IO Driver list.
6. Enter the **IO Physical Address**. The address is:
   0.0.ZZ is the module I/O number (1-48)
7. Click **[OK]**.

To associate a variable with the pulse input:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** from the **Type** list.
4. Select **Input** by clicking the check box.
5. Select **Processor_2010** from the **IO Driver** list.
6. Select **WORD** from the **IO Size** list.
7. Enter the **IO Physical Address.** The address is:
   0.Y.1 Y is the type of I/O (1 = Count, 2 = Frequency)
8. Click **[OK]**.

PPC-2010 Processor Module Addressing Example

To create a variable named My_Dig_Out that sets the value of the 23rd output on a PPC-2010 module, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



NOTE! ***The PPC-2010's module address is 0 (zero).***

Analog_In_202x

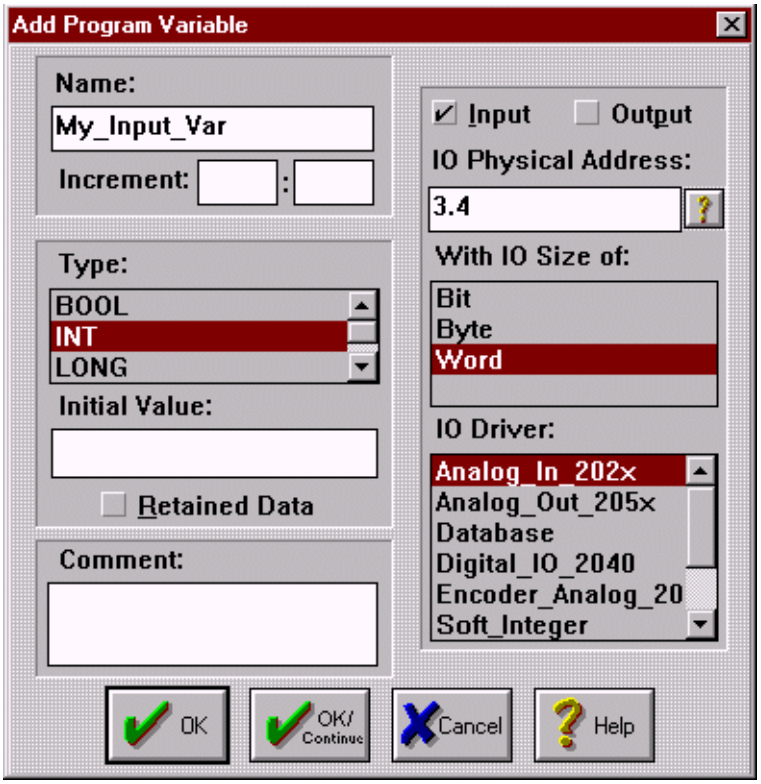
The Analog_In_202x driver supports the inputs on the PPC-202x series Analog In modules:

To associate a variable with an analog input:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** from the **Type** list.
4. Select **Input** by clicking the check box.
5. Select **Analog_In_202x** from the **IO Driver** list.
6. Select **WORD** from the **IO Size** list.
7. Enter the **IO Physical Address.** The address is:
   X.Z X is the module address (1-4)
   Z is the module I/O number (1-32).
8. Click **[OK]**.

PPC-2020 Analog In Module Addressing Example

To create a variable named My_Input_Var that reads the value of the fourth thermocouple on a PPC-2021 module with its rotary address switch set to 3, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Encoder_Analog_2030

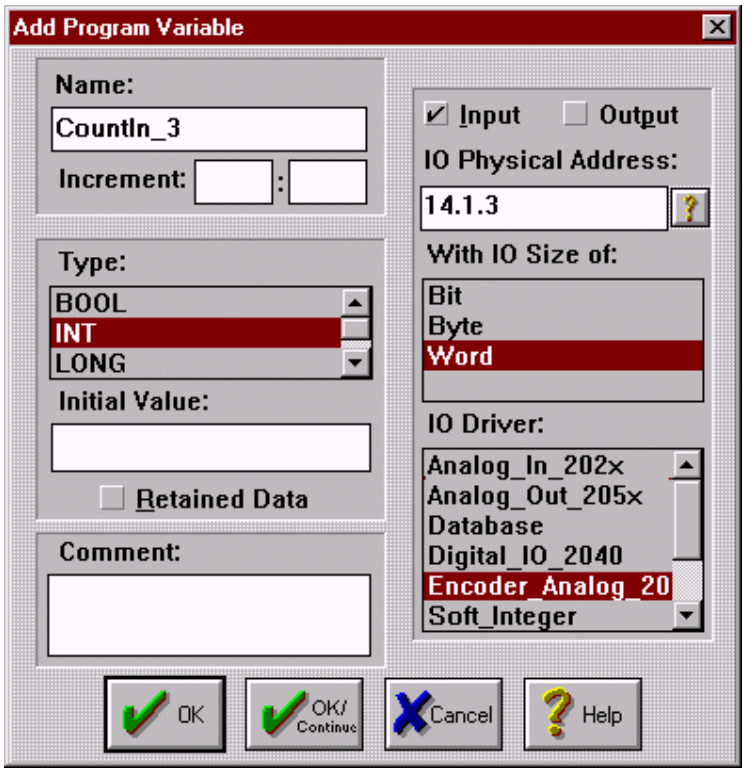
The Encoder_Analog_2030 driver supports the inputs and outputs on PPC-2030 Encoder In Analog Out Module.

To associate a variable with an encoder input or an analog output:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** from the **Type** list.
4. Select **Input** or **Output** by clicking the corresponding check box.
5. Select **Encoder_Analog_2030** from the **IO Driver** list.
6. Select **WORD** from the **IO Size** list.
7. Enter the **IO Physical Address**. The address is:
   X.Y.Z X is the module address (11-14)
   Y is the type of I/O (1 = Count, 2 = Frequency, 3 = Analog Out)
   Z is the module I/O number (1-4)
8. Click **[OK]**.

PPC-2030 Encoder In Analog Out Addressing Example

To create a variable named CountIn_3 that reads the count on the third input on a PPC-2030 module with its rotary address switch set to 14, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Digital_IO_2040

The Digital_IO_2040 driver supports the inputs and outputs on the PPC-2040 Digital IO Module.

To associate a variable with a digital I/O point:

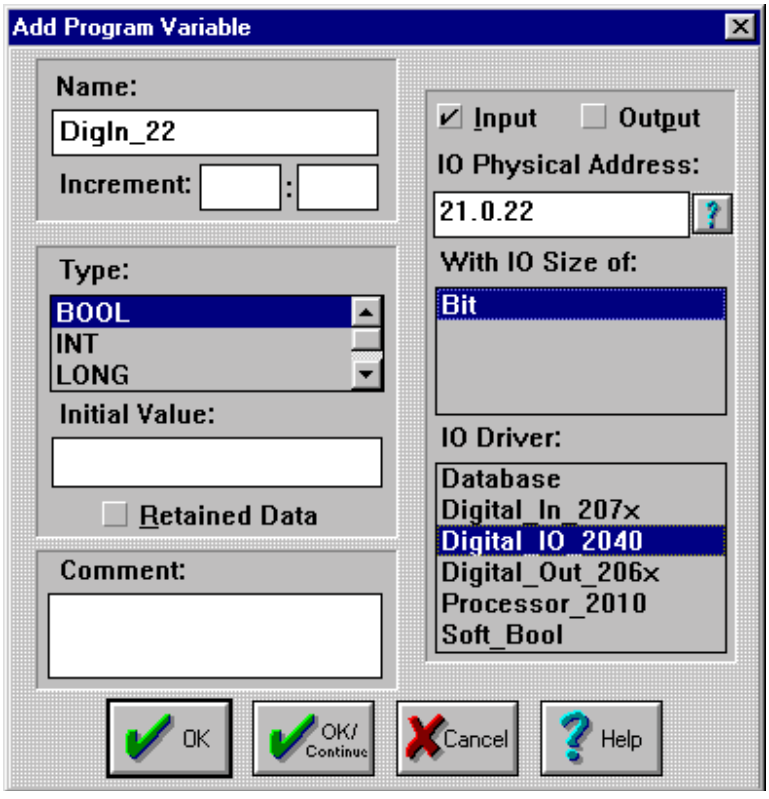
1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **BOOL** from the **Type** list.
4. Select **Input** or **Output** by clicking the corresponding check box.
5. Select **Digital_IO_2040** from the **IO Driver** list.
6. Enter the **IO Physical Address**. The address is:
   X.0.Z X is the module address (21-26)
   Z is the module I/O number (1-32).
7. Click **[OK].**

To associate a variable with a counter input:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** from the **Type** list.
4. Select **Input** by clicking the check box.
5. Select **Digital_IO_2040** from the **IO Driver** list.
6. Select **WORD** from the **IO Size** list.
7. Enter the **IO Physical Address**. The address is:
   X.Y.Z X is the module address (21-26)
   Y is the type of I/O (1 =Count, 2 =Frequency)
   Z is the module I/O number (1-2).
8. Click **[OK].**

PPC-2040 Digital I/O Addressing Example

To create a variable named DigIn_22 that reads the state of the 22nd input on a PPC-2040 module with its rotary address switch set to 21, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Analog_Out_205x

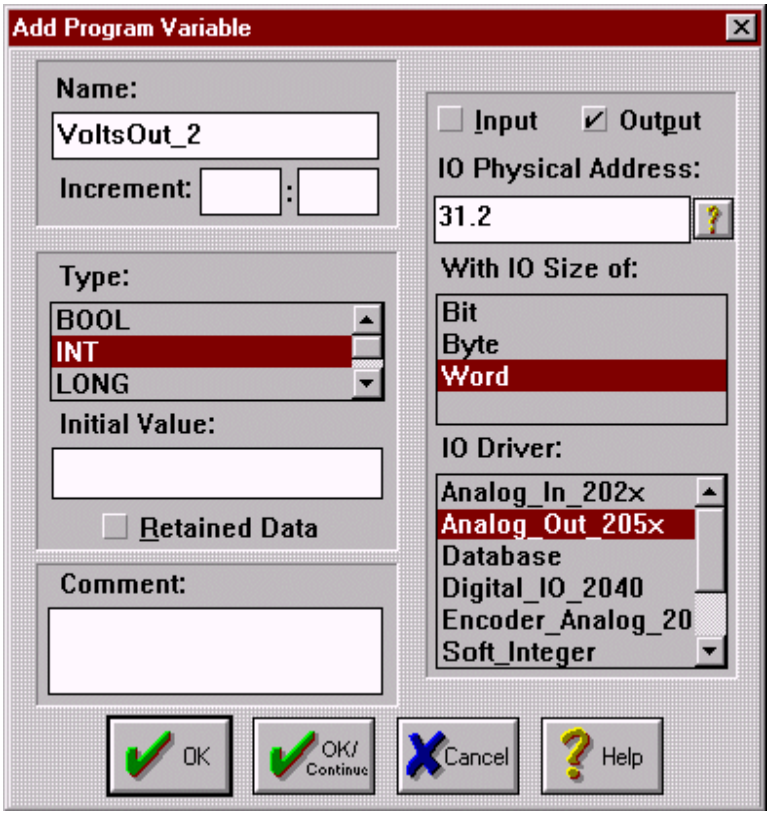
The Analog_Out_205x driver supports the outputs on the PPC-205x series Analog Output modules.

To associate a variable with an analog output:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** from the **Type** list.
4. Select **Output** by clicking the check box.
5. Select **Analog_Out_205x** from the **IO Driver** list.
6. Select **WORD** from the **IO Size** list.
7. Enter the **IO Physical Address.** The address is:
   X.Z X is the module address (31-34)
   Z is the module I/O number (1-8)
8. Click **[OK].**

PPC-205x Analog Out Addressing Example

To create a variable named VoltsOut_2 that sets the level of the second output on a PPC-205x module with its rotary address switch set to 31, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Digital_Out_206x

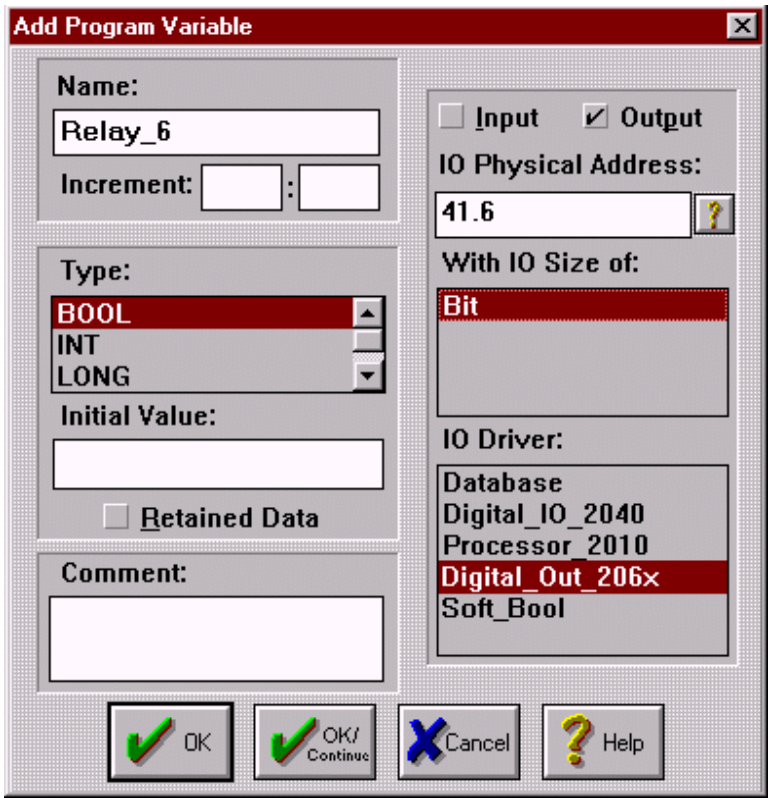
The Digital_Out_206x driver supports the relay outputs on the PPC-206x Digital Out series modules.

To associate a variable with a relay output:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **BOOL** from the **Type** list.
4. Select **Output** by clicking the check box.
5. Select **Digital_Out_206x** from the **IO Driver** list.
6. Select **BIT** from the **IO Size** list.
7. Enter the **IO Physical Address.** The address is:
   X.Z X is the module address (41-46)
   Z is the module I/O number (1-16)
8. Click **[OK].**

PPC-206x Digital Out Addressing Example

To create a variable named Relay_6 that sets the state of the 6th output on a PPC-206x module with its rotary address switch set to 41, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Digital_In_207x

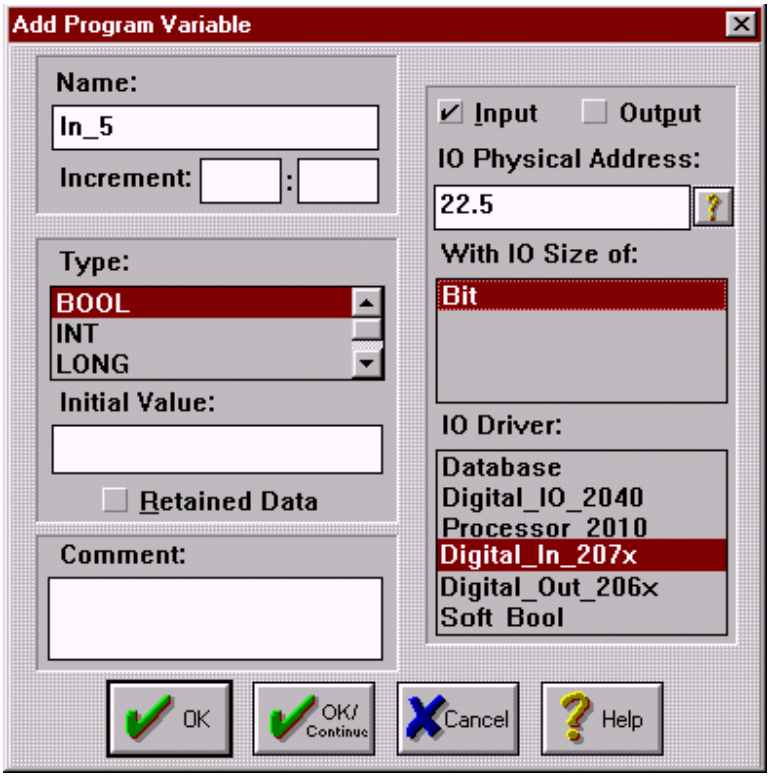
The Digital_In_207x driver supports the inputs on the PPC-207x Digital In series modules.

To associate a variable with a digital input:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **BOOL** from the **Type** list.
4. Select **Input** by clicking the check box.
5. Select **Digital_In_207x** from the **IO Driver** list.
6. Enter the **IO Physical Address.** The address is:
   X.Z X is the module address (51-54)
   Z is the module I/O number (1-16)
7. Click **[OK].**

PPC-207x Digital In Addressing Example

To create a variable named In_5 that examines the state of the fifth input on a PPC-207x module with its rotary address switch set to 52, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Setpoint

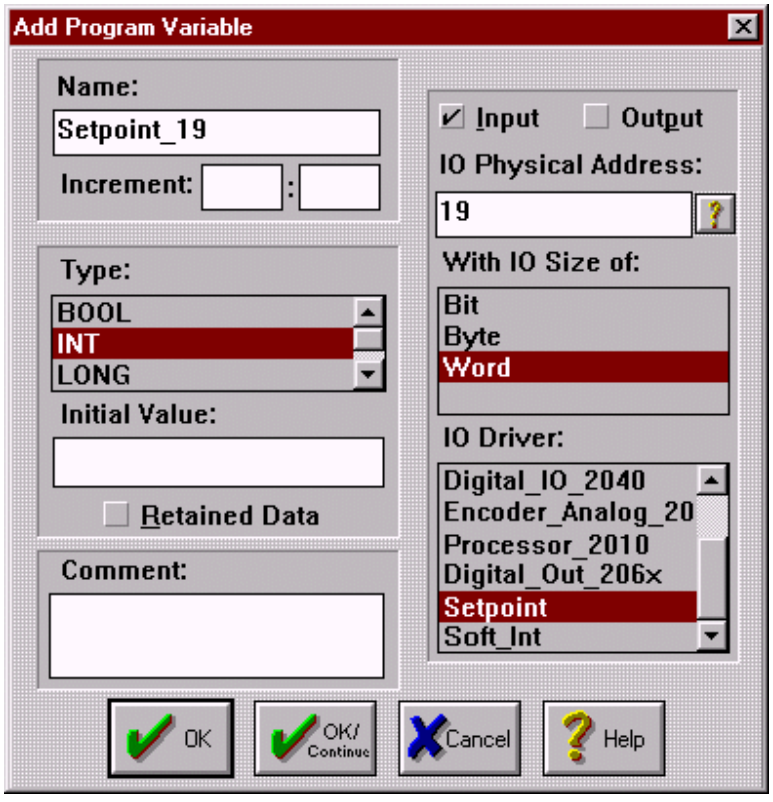
A channel's setpoint can be used in logic programs by creating a variable in the logic program that points to this controller parameter as if it were physical I/O. This allows the logic program to examine or set value of a setpoint. The I/O address for a setpoint is constructed from the channel number only.

To associate a variable with a loop setpoint:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** from the **Type** list.
4. Select **Input** or **Output** by clicking the corresponding check box.
5. Select **Setpoint f**rom the **IO Driver** list.
6. Select **WORD** from the **IO Size** list.
7. In the **IO Physical Address** field, enter the channel number (1-48).
8. Click **[OK].**

Setpoint Addressing Example

To create a variable named Setpoint_19 that reads the value of the Setpoint of channel 19, set the options in the **Add Program/ Resource Variable** dialog box as illustrated below.



Soft_Input

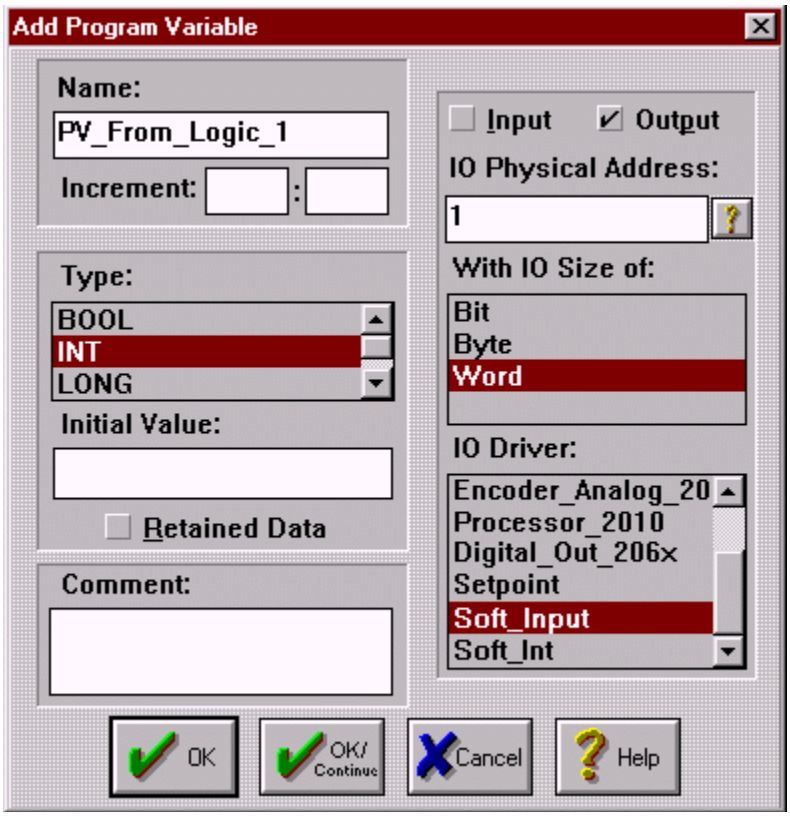
A logic program can generate an analog input to a closed-loop control channel. To do so, output a value to one of the 50 Soft Input registers and select that register as the PV Source for the channel.

To associate a variable with a soft input:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** from the **Type** list.
4. Select **Output** by clicking the check box.
5. Select **Soft_Input** from the **IO Driver** list.
6. Select **WORD** from the **IO Size** list.
7. In the **IO Physical Address** field, enter the Soft Input number (1-50).
8. Click **[OK].**

Soft Input Addressing Example

To create a variable named PV_From_Logic_1 that sets the value of the first Soft Input, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Soft_Bool

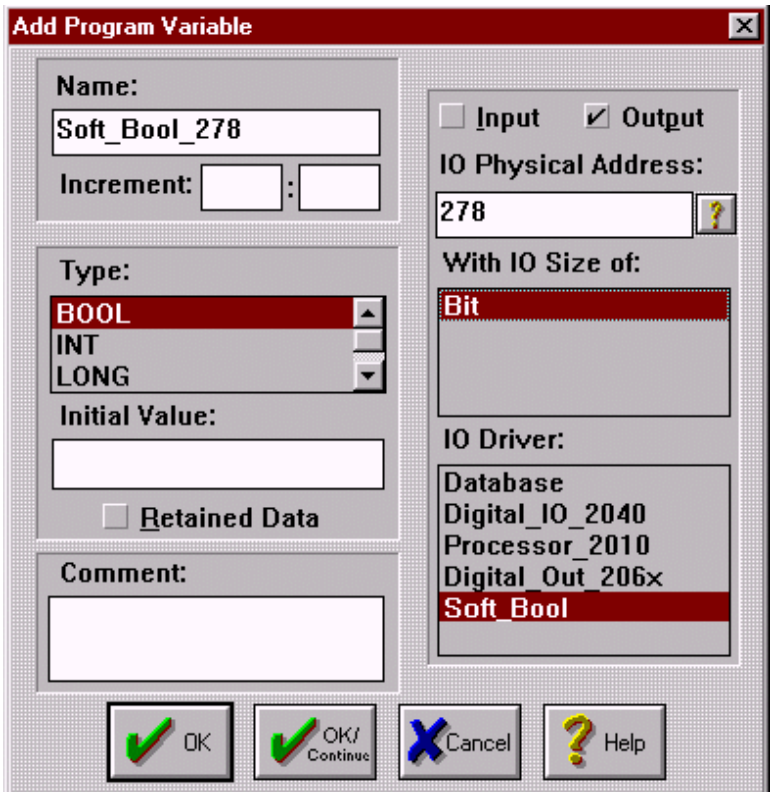
A Soft Bool can be used in logic programs by creating a variable in the logic program that points to the Soft Bool as if it were physical I/O. This allows the logic program to examine and set values of Soft Bools. For example, an HMI program could allow a user to set a software switch that is read by and affects the operation of a logic Program. The I/O address for Soft Bools is the Soft Bool number.

To associate a variable with a Soft Bool:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name.**
3. Select **BOOL** from the **Type** list.
4. Select **Input** or **Output** by clicking the corresponding check box.
5. Select **Soft_Bool** from the **IO Driver** list.
6. In the **IO Physical Address** field, enter the Soft Bool number (1-1000).
7. Click **[OK].**

Soft Bool Addressing Example

To create a variable named Soft_Bool_278 that sets the value of Soft Bool 278, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Soft_Int

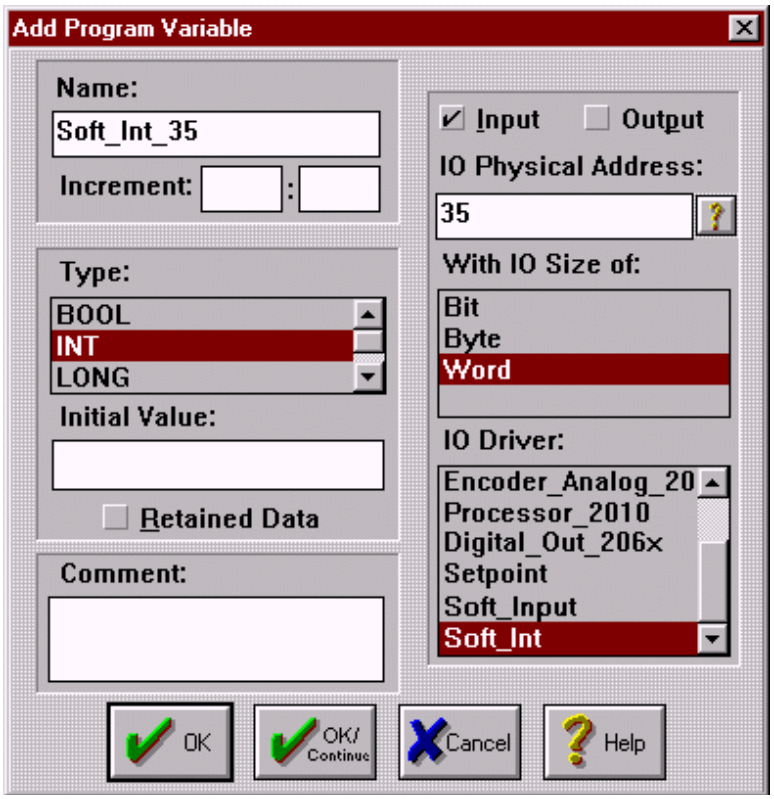
A Soft Int can be used in logic programs by creating a variable in the logic program that points to the Soft Int as if it were physical I/O. This allows the logic program to examine and set values of Soft Ints. For example, a Soft Int could be used to allow a user to set a timer preset in a logic Program. The I/O address for a Soft Int is the Soft Int number.

To associate a variable with a Soft Int:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name.**
3. Select **INT** from the **Type** list.
4. Select **Input** or **Output** by clicking the corresponding check box.
5. Select **Soft_Int** from the **IO Driver** list.
6. Select **WORD** from the **IO Size** list.
7. In the **IO Physical Address** field, enter the Soft Int number (1-100).
8. Click **[OK].**

Soft Int Addressing Example

To create a variable named Soft_Int_35 that reads the value of Soft Int 35, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Database

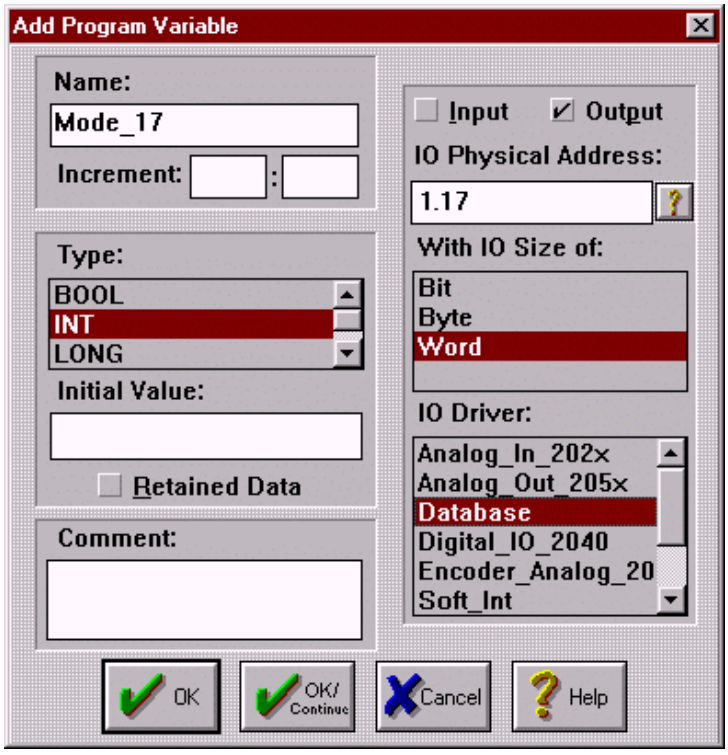
The Database I/O Driver is used to examine or set values for any closed-loop control parameter not supported by one of the other drivers. The specific parameter value is addressed with the parameter number and its database offset.

To associate a variable with a value in the database:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name.**
3. Select **INT** or **BOOL** from the **Type** list.
4. Select **Input** or **Output** by clicking the corresponding check box.
5. Select **Database** from the **IO Driver** list.
6. Select **WORD** or **BIT** from the **IO Size** list.
7. Enter the **IO Physical Address.** The address is:
   #.D # is the parameter number
   D is the database offset
8. Click **[OK]**.

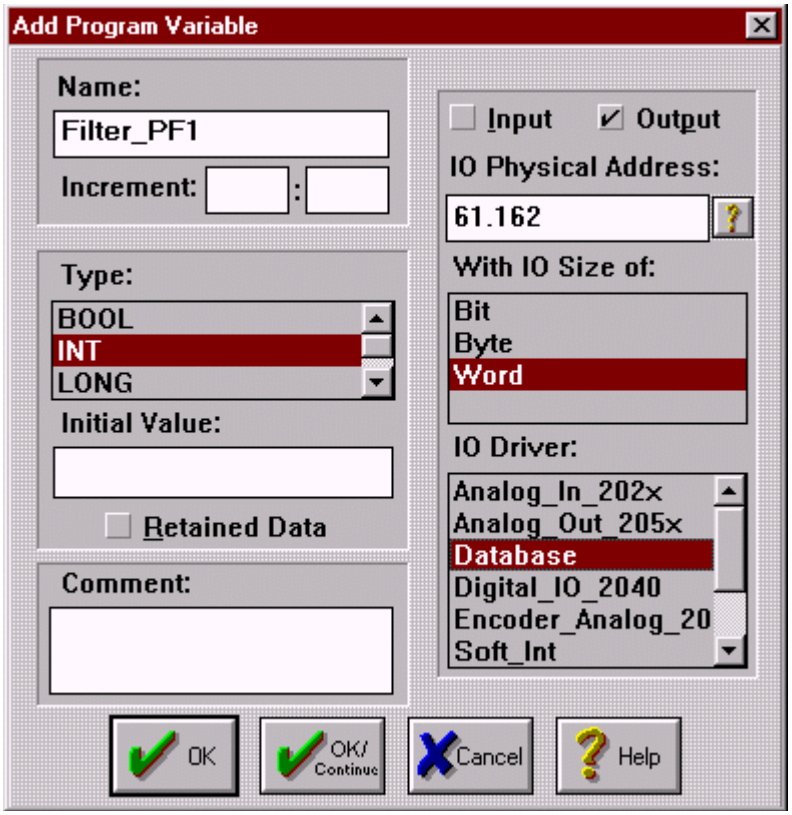
Channel Parameter Example

To create a variable named Mode_17 that sets the control mode for channel 17, set the options in the **Add Program/Resource Variable** dialog box as illustrated below. The parameter number for control mode is 1. Control mode is a channel parameter, so the database offset is the channel number, 17.



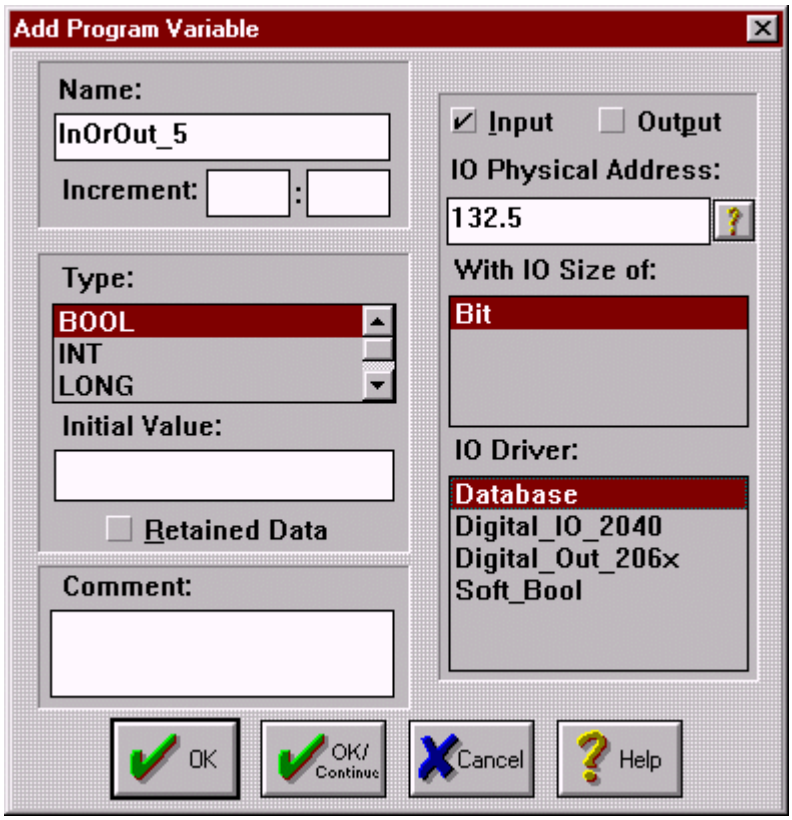
Analog Input Parameter Example

To create a variable named Filter_PF1 that sets the input filter for the frequency input on the processor module, set the options in the **Add Program/Resource Variable** dialog box as illustrated below. The parameter number for input filter is 61. The database offset, 162, is found in the PPC-2000 User's Guide.



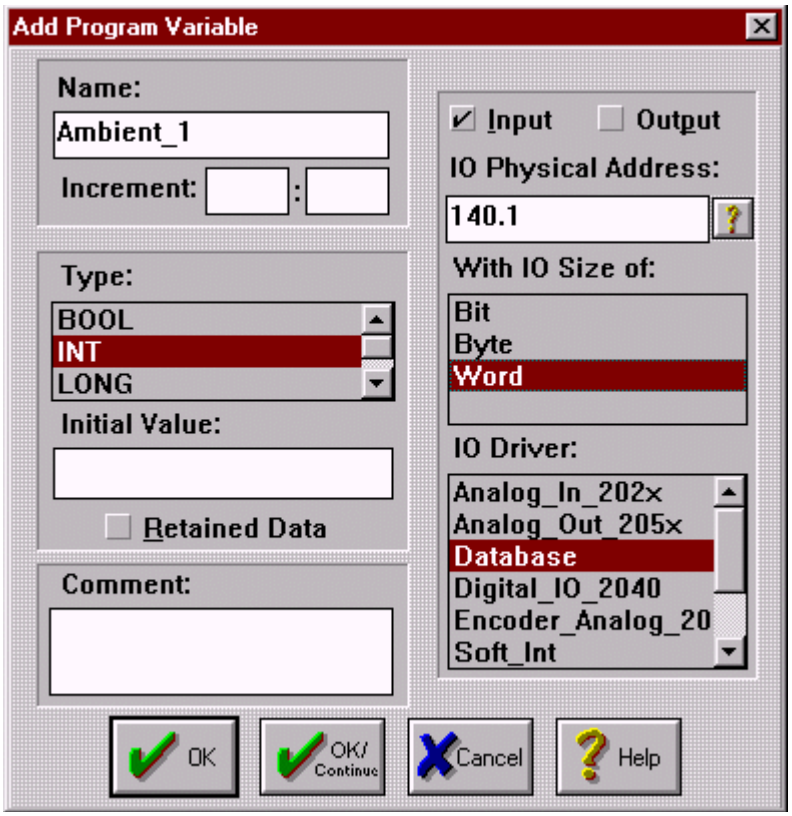
Digital I/O Parameter Example

To create a variable named InOrOut_5 that examines the direction parameter setting for the fifth I/O point on the processor module, set the options in the **Add Program/Resource Variable** dialog box as illustrated below. The parameter number for direction is 132. The database offset, 5, is found in the PPC-2000 User's Guide.



Global Parameter Example

To create a variable named Ambient_1 that reads the ambient temperature at the first cold junction compensation sensor on the first Analog In module, set the options in the **Add Program/ Resource Variable** dialog box as illustrated below. The parameter number for ambient temperature is 140. The database offset, 1, is found in the PPC-2000 User's Guide.



CPC400 IO Drivers

IO Driver Choice

The IO Driver allows the logic program to examine or set an I/O value or other value stored in the database. Choose the appropriate driver for the value which you want to access. *Table 12.10* describes what data can be accessed with each IO Driver.

Table 12.10 Data Accessed by the IO Drivers

IO Driver	Data Accessed
CPC400_PV	Process variable (scaled analog input) value for any channel.
CPC400_Digital_In	State of any digital input.
CPC400_Digital_Out	State of any digital output.
Setpoint	Setpoint value of any of the channels.
Soft_Bool	The value of any soft bool register. Typically used to exchange information between a logic program and an HMI program or operator interface panel.
Soft_Input	Value of any of the soft inputs. Typically used for a logic program to output a value to the input of a closed-loop control channel.
Soft_Int	The value of any soft int register. Typically used to exchange information between a logic program and an HMI program or operator interface panel.
Database	The value of any closed-loop control parameter that is not supported by another IO Driver.

IO Physical Addresses

The IO Physical Address syntax depends on the IO Driver. The CPC400 Series User's Guide lists the addresses for each I/O point and parameter. In general there are three types of I/O drivers. *Table 12.11* lists the IO Drivers in the categories that characterize the IO Physical Addresses used with them.

Table 12.11 Categories of IO Drivers

Category	Drivers
Hardware Specific	CPC400_PV CPC400_Digital_Out CPC400_Digital_In
Parameter Specific	Setpoint Soft_Input Soft_Bool Soft_Int
Database	Database

The following sections describe the address syntax used with each category of IO Driver.

Hardware Specific Drivers

The hardware IO drivers are used to access digital I/O states and process variable values. The specific IO value is addressed by number. *Table 12.12* lists the hardware specific drivers and the corresponding database offset ranges.

Table 12.12 Addresses Used for Hardware Specific Drivers

IO Driver	IO Physical Addresses
CPC400_PV	Channel Number (1 to 5) CPC404, (1 to 9) CPC408
CPC400_Digital_Out	Digital Output Number (1 to 34)
CPC400_Digital_In	Digital Input Number (1-8)

Parameter Specific Drivers

The parameter specific IO Drivers (Setpoint, Soft_Input, Soft_Bool, and Soft_Int) are used to read or set values for specific parameters. The specific parameter value is addressed with its database offset.

Table 12.13 lists the parameter specific drivers and the corresponding database offset ranges.

Table 12.13 Addresses Used with Parameter Specific IO Drivers

IO Driver	Database Offset
Setpoint	Channel Number (1 to 5) CPC404, (1 to 9) CPC408
Soft_Input	Soft Input Number (1-9)
Soft_Bool	Soft Bool Number (1-256)
Soft_Int	Soft Int Number (1-100)

Database Driver

The Database IO Driver is used to examine or set values for any closed-loop control parameter such as Control Mode that is not supported by one of the other drivers. The Database driver allows access to all controller parameters. Other drivers are provided for faster access to specific parameters. Of course, controller parameters that are read-only cannot be adjusted. The specific parameter value is addressed with the parameter number and its database offset. See *Table 12.14 on page 210* for a description of this syntax.

Table 12.14 Address Syntax for Database Driver

Syntax	Used with this IO Driver
#.D # = Parameter Number D = Database Offset	Database

Two types of parameters are accessed using the Database I/O driver: Channel and Global. The following sections describe how to determine the database offsets to be used to access each of these types of parameters.

Parameter Number

The parameter number is the numeric identifier used to address a closed-loop control parameter. Each parameter in the CPC400 has a parameter number. See the CPC400 manual for a list of parameter numbers and descriptions.

Database Offset

The database offset is the location of a particular value relative to the first value of that parameter. For example, each channel has a proportional band setting. If you want to set the proportion band for channel 6, the address will include the database offset 6.

Channel Parameters

The database offsets associated with channel parameters are the channel numbers. Refer to the CPC400 Series User's Guide for a complete description of channel parameters and a list of channel parameter numbers.

Global Parameters

Global parameter offsets are dependent on the global parameter. See the CPC400 Series User's Guide for a complete list of global parameter numbers and their corresponding offsets.

Using the IO Drivers

The following sections describe and illustrate the use of each IO Driver.

CPC400_PV

The CPC400_PV driver allows a logic program to access the process variables (PV) for each channel.

To read a PV with a logic program variable:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.

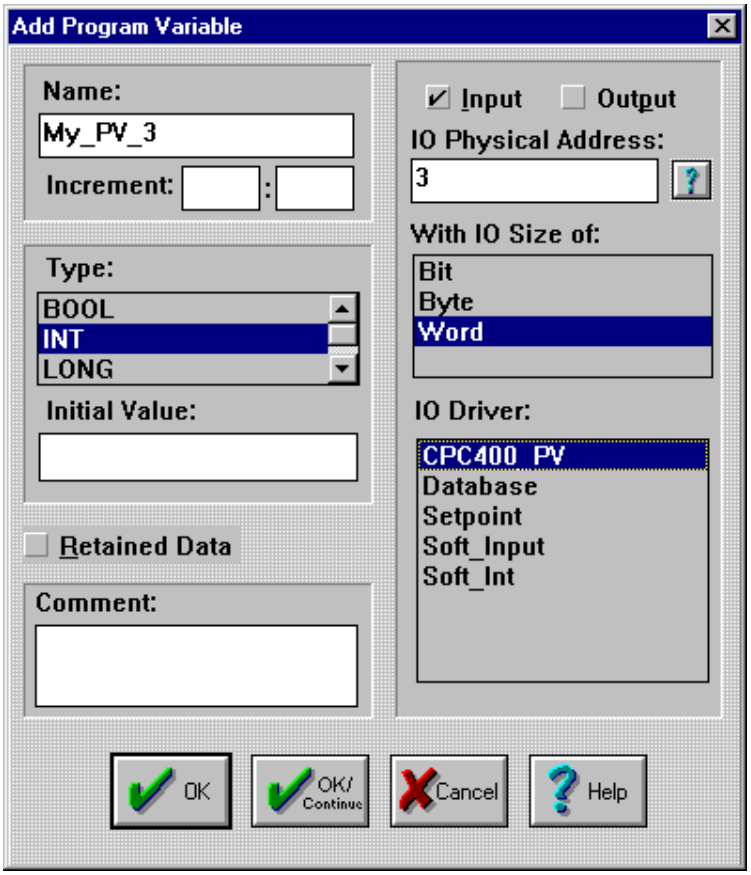
3. Select **INT** from the **Type** list.
4. Select **Input** by clicking the check box.

NOTE ! ***A variable using the CPC400_PV driver should always be an input an never an output. To set the input to a loop with a logic program, use a soft input.***

5. Select **CPC400_PV** from the **IO Driver** list.
6. Select **WORD** from the **IO Size** list.
7. In **IO Physical Address** field, enter the channel number (1 to 5 for CPC404; 1 to 9 for CPC408).
8. Click **[OK]**

CPC400_PV Addressing Example

To create a variable named My_PV_3 that reads the value of the channel 3's process variable, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



CPC400_Digital_In

The CPC400_Digital_In driver allows a logic program to access the state of each of the digital inputs.

To read a digital input with a logic program variable:

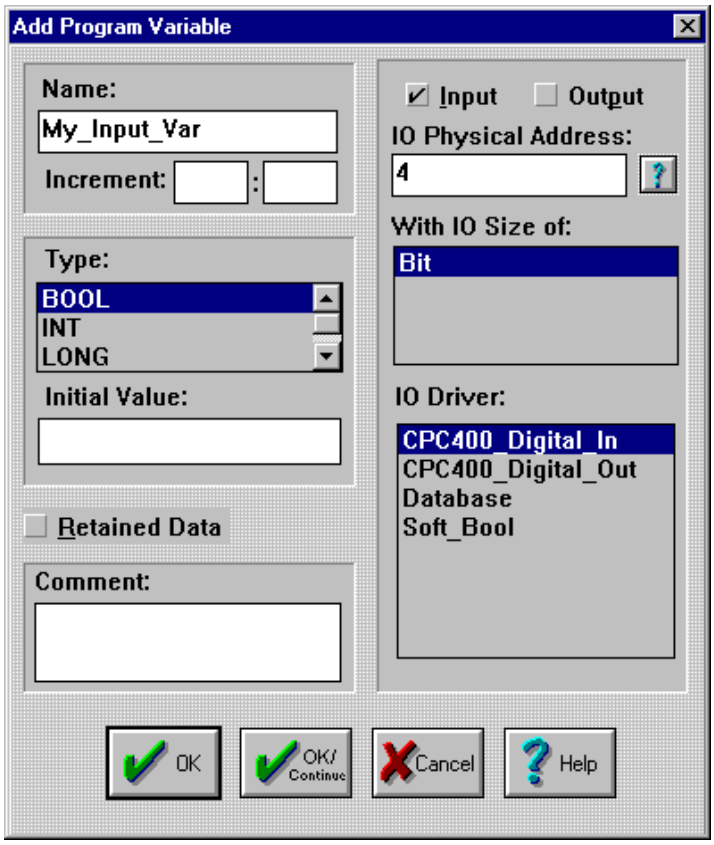
1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **BOOL** from the **Type** list.
4. Select **Input** by clicking the check box.

NOTE! ***A variable using the CPC400_Digital_In driver should always be an input and never an output.***

5. Select **CPC400_Digital_In** from the **IO Driver** list.
6. In **IO Physical Address** field, enter the digital input number (1 to 8).
7. Click **[OK]**.

CPC400_Digital_In Addressing Example

To create a variable named My_Input_Var that reads the value of the 4th digital input on a CPC400, set the options in the **Add Program/Resource Variable** dialog box as listed below.



CPC400_Digital_Out

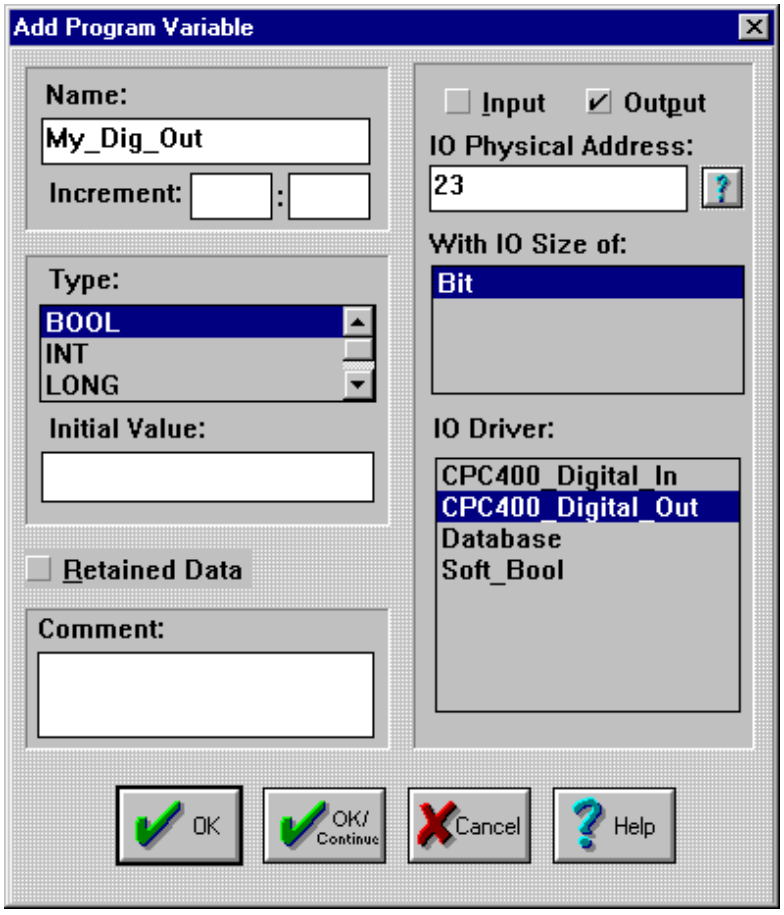
The CPC400_Digital_Out allows a logic program to access the state of each of the digital outputs.

To read or set a digital output with a logic program variable:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **BOOL** from the **Type** list.
4. Select **Input** or **Output** by clicking the corresponding check box.
5. Select **CPC400_Digital_Out** from the **IO Driver** list.
6. In the **IO Physical Address** field, enter the digital output number (1 to 34).
7. Click **[OK]**

CPC400_Digital_Out Addressing Example

To create a variable named My_Dig_Out that sets the value of the 23rd output on a CPC400, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Setpoint

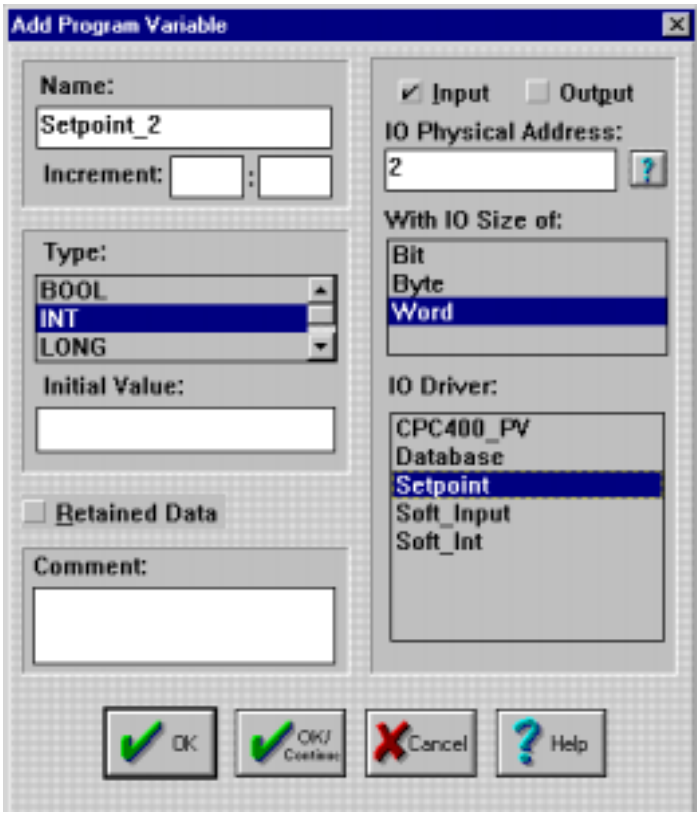
A setpoint can be used in a logic program by creating a variable in the logic program that points to the setpoint parameter as if it were physical I/O. This allows the logic program to examine or set the value of a setpoint. The I/O address for Setpoint is constructed from the channel number only.

To read or set a setpoint with a logic program variable:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** from the **Type** list.
4. Select **Input** or **Output** by clicking the corresponding check box.
5. Select **Setpoint** from the **IO Driver** list.
6. Select **WORD** as the **IO** Size.
7. In the **IO Physical Address** field enter the channel number (1 to 5 for CPC204; 1 to 9 for CPC208).
8. Click **[OK]**.

Setpoint Addressing Example

To create a variable named Setpoint_2 that reads the value of the setpoint of channel 2, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Soft_Input

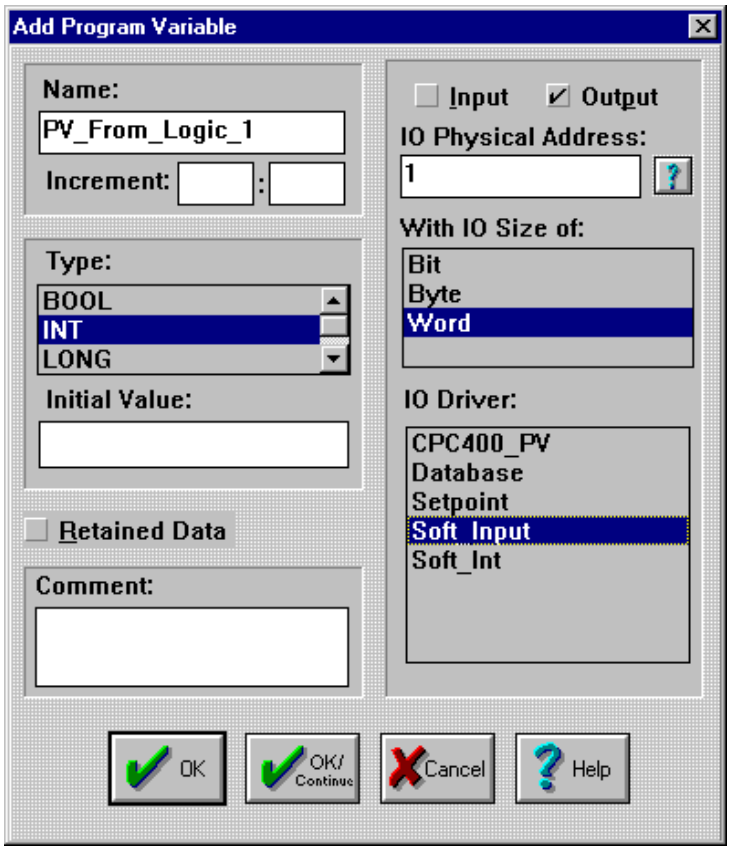
A logic program can generate an analog input to a closed-loop control channel. To do so, output a value to one of the **Soft Input** registers and enable the soft input for use on the channel.

To set a Soft Input with a logic program variable:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** from the **Type** list.
4. Select **Output** by clicking the check box.
5. Select **Soft_Input** from the **IO Driver** list.
6. Select **WORD** as the **IO Size**.
7. In the **IO Physical Address** field enter the Soft Input number (1-9).
8. Click **[OK]**.

Soft Input Addressing Example

To create a variable named PV_From_Logic_1 that sets the value of the first soft input, set the options in the Add Program/Resource Variable dialog box as illustrated below.



Soft_Bool

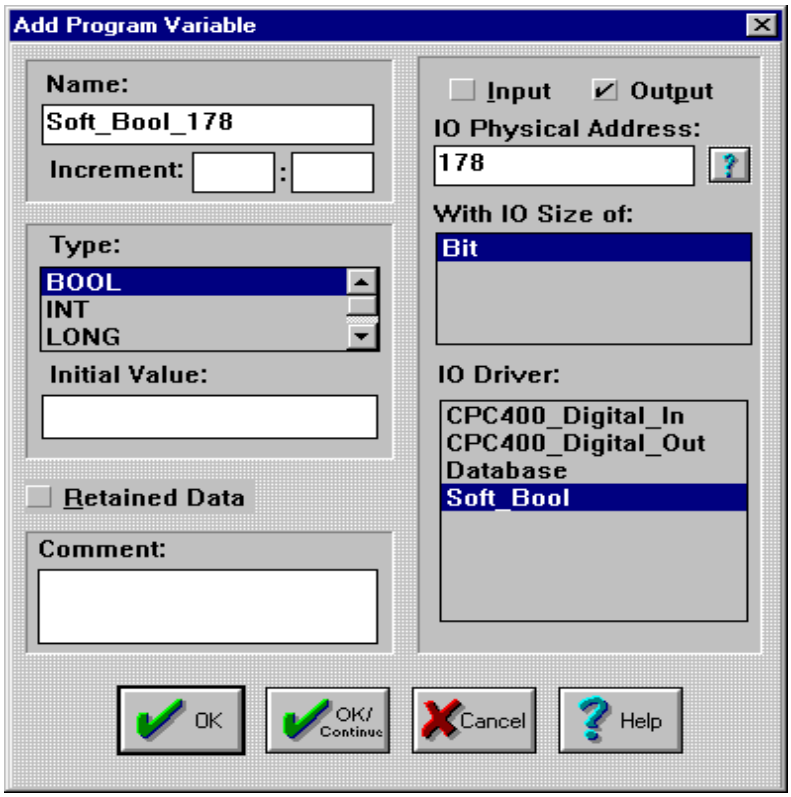
A Soft Bool can be used in logic programs by creating a variable in the logic program that points to the Soft Bool as if it were physical I/O. This allows the logic program to examine and set values of Soft Bools. The I/O address for Soft Bools is the Soft Bool number.

To read or set a Soft Bool with a logic program variable:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **BOOL** from the Type list.
4. Select **Input** or **Output** by clicking the corresponding check box.
5. Select **Soft_Bool** from the **IO Driver** list.
6. In the **IO Physical Address** field enter the Soft Bool number (1-256).
7. Click **[OK]**

Soft Bool Addressing Example

To create a variable named Soft_Bool_178 that sets the value of soft bool 178, set the options in the **Add Program/ Resource Variable** dialog box as illustrated below.



Soft_Int

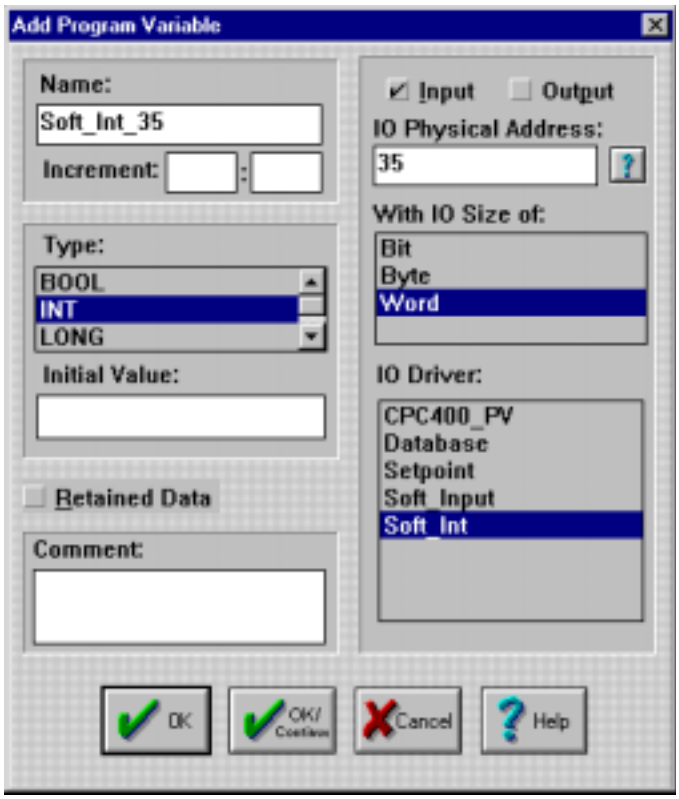
A Soft Int can be used in logic programs by creating a variable in the logic program that points to the Soft Int as if it were physical I/O. This allows the logic program to examine and set values of Soft Ints. The I/O address for a Soft Int is the Soft Int number.

To read or set a Soft Int with a logic program variable:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** from the **Type** list.
4. Choose **Input** or **Output** by clicking on the corresponding check box.
5. Select **Soft_Int** from the **IO Driver** list.
6. Select **WORD** as the **IO Size**.
7. In the **IO Physical Address** field, enter the **Soft Int** number (1-100).
8. Click **[OK]**

Soft Int Addressing Example

To create a variable named Soft_Int_35 that reads the value of soft int 35, set the options in the **Add Program/Resource Variable** dialog box as illustrated below.



Database

The Database I/O Driver is used to examine or set values for any closed-loop control parameter not supported by one of the other drivers. The specific parameter value is addressed with the parameter number and its database offset.

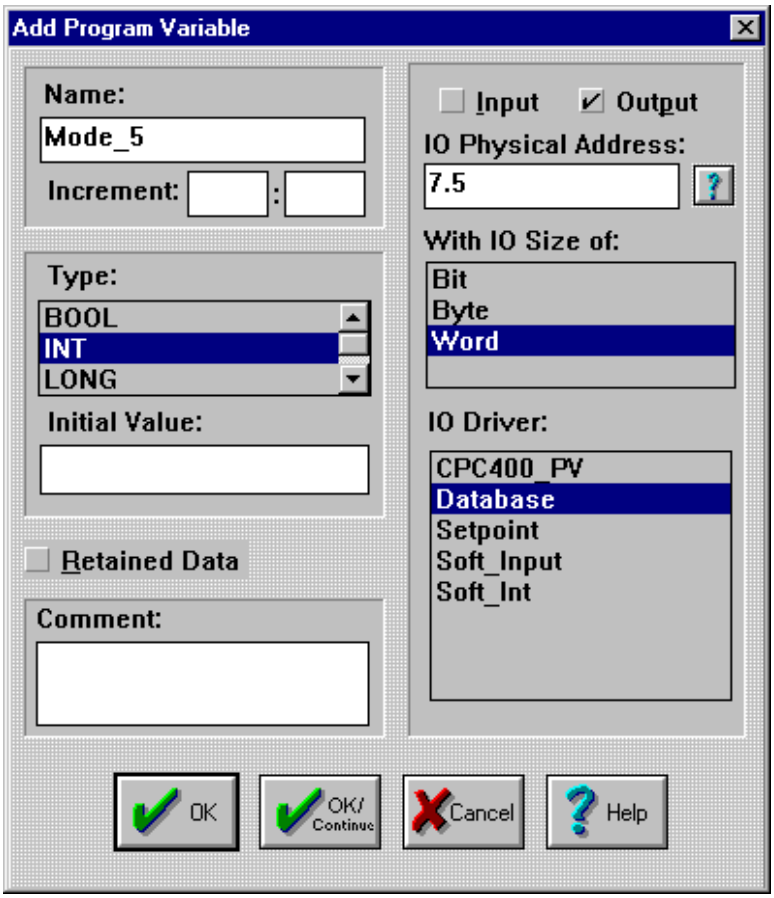
To associate a variable with a value in the database:

1. Open the **Add Program/Resource Variable** dialog box.
2. Enter a unique variable **Name**.
3. Select **INT** or **BOOL** from the **Type** list.
4. Choose **Input** or **Output** by clicking on the corresponding check box.
5. Select **Database** from the **IO Driver** list.
6. Select **WORD** or **BIT** as the **IO Size**.
7. Enter the **IO Physical Address**. The address is:

   #.D #is the parameter number
   D is the database offset
8. Click **[OK]**.

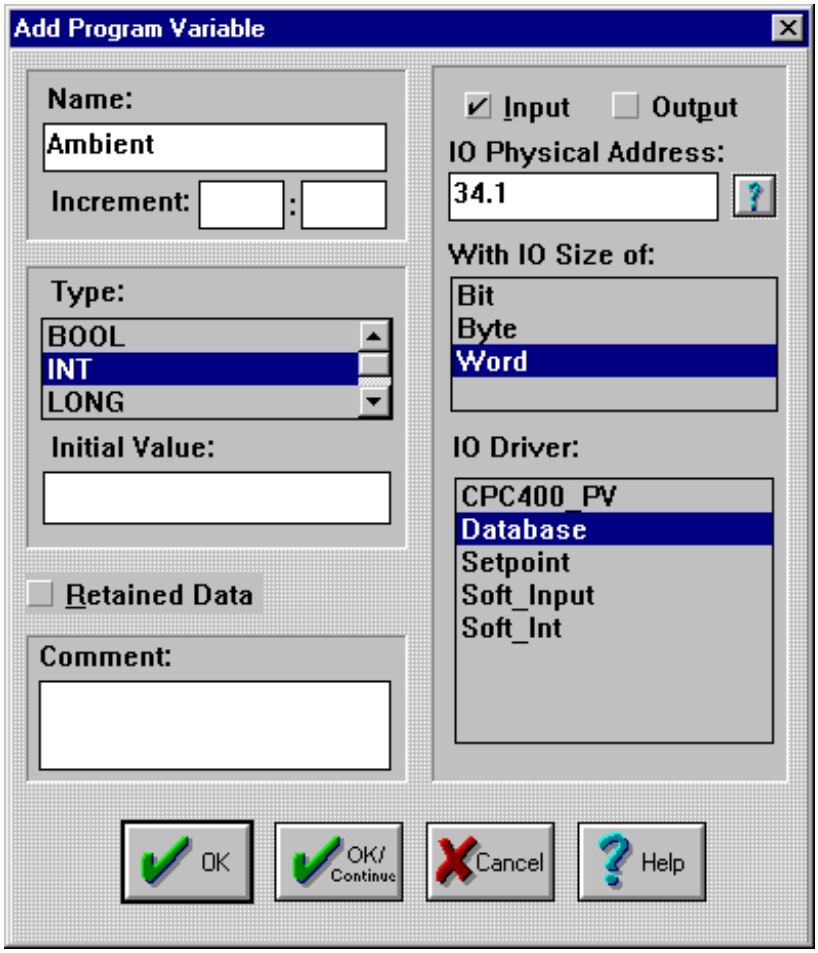
Channel Parameter Example

To create a variable named Mode_5 that sets the control mode for channel 5, set the options in the **Add Program/ Resource Variable** dialog box as illustrated below. The parameter number for control mode is 7. Control mode is a channel parameter, so the database offset is the channel number, 5.



Global Parameter Example

To create a variable named Ambient that reads the ambient temperature at the cold junction compensation sensor, set the options in the **Add Program/Resource Variable** dialog box as illustrated below. The parameter number for ambient temperature is 34. The database offset, 1, is found in the CPC400 User's Guide.



13

Compiler Setup

LogicPro uses the Borland C++ 4.0 compiler, both for simulation and for logic programs down loaded to a controller.

Setting up the Compiler

The compiler configuration tells *LogicPro* the location (path) of the compiler, linker and library executables along with the Include and Library paths.

Installation sets up the compiler paths. Under most circumstances, the default settings are adequate. If the paths need to be modified, use the following procedure to define the compiler paths:

1. Select the **Options** menu.
2. Select **Resource Compiler Selection**. This will cause the **Resource Compiler Selection** dialog box to open, as shown in *Figure 13.1*.

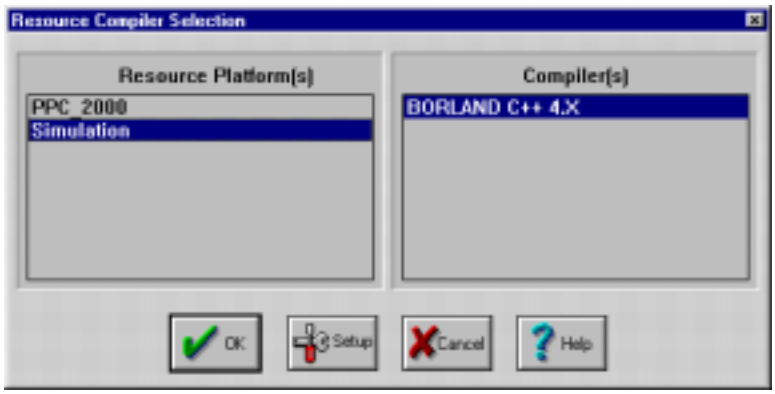


Figure 13.1 Simulation Compiler

3. Click **[Setup]**. This causes the **Compiler Paths** dialog box to open, as shown in *Figure 13.2*.

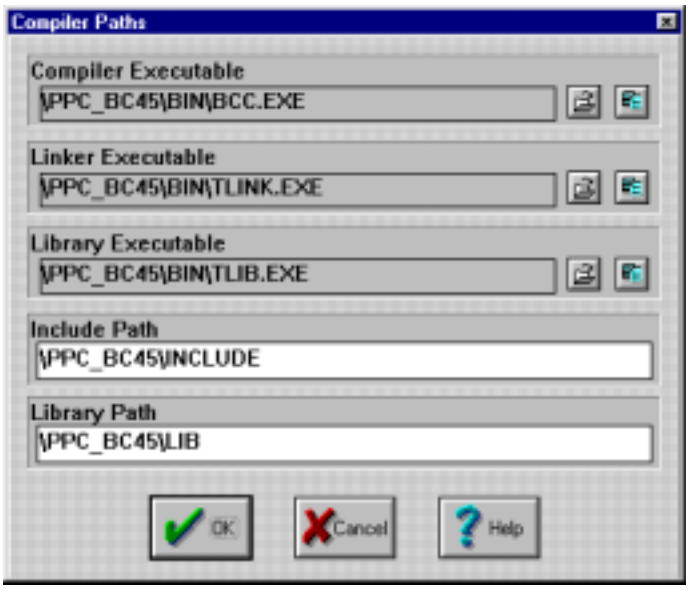


Figure 13.2 Compiler Paths

4. Click the **[Open]** button or the **[Copy Field]** button to the right of the **Compiler, Linker** or **Library** field you want to configure. This will cause the corresponding **Select Compiler** dialog box to open. See *Figure 13.3*.

Figure 13.3 Select Borland Compiler

5. Select a new path for the executable from the **Files** list box.

6. Repeat from *Step 4* as necessary for the **Linker** and **Library**.

7. You may enter values into the **Include Path** and **Library Path** fields by typing them directly into the dialog box.

If the same path exists for the **Compiler, Linker** and **Library** you can update all of the path fields by clicking the **[Copy Field]** button.

14

Simulation

LogicPro allows you to simulate your control logic program without the need for a controller. After you have completed writing your application program and before you load to a target resource, Simulation allows you to test it for problems. This is a simple process and does not require you to make changes to any of the I/O assignments, resource definitions, or resource setups.

Building a Simulation causes *LogicPro* to generate and compile code to create a Windows Dynamic Link Library (DLL). Then, when you run the Simulation *LogicPro* communicates with this DLL instead of your target resource. This is a very powerful tool for testing, debugging and optimizing the performance of your program.

The advantages of using *LogicPro* simulation are:

- Same logic code
- No I/O

While using the Simulation mode, *LogicPro* allows you to accelerate the time base by adjusting the Timer Ticks setting. You can control the number of times your logic executes by adjusting the number of scans setting. Finally, you can change individual variable settings. This gives you control over all aspects of the simulation.

Building a Program for Simulation

The *LogicPro* Simulation feature allows you to scan, step through and view individual processes in several ways from start to finish. *LogicPro* allows you to scan your program one time, ten times, one hundred times or in a continuous (free running) mode. You can also toggle all of your Boolean variables, change constants and assign values to all variables. This allow you to fully test your program and simulate various operating conditions prior to downloading it to your resource.

When you are finished with your simulation you will need to compile your program again before it is ready to download to your target resource.

To start simulation:

1. Select the **Simulation** menu.
2. Select **Build All and Run**.

   The **Compiler** dialog box will open, as *LogicPro* begins to compile the program in the active window. See *Figure 14.1 on page 227*.

   Next, a DOS window will open as your Borland C++ application compiles your code, see *Figure 14.2 on page 227*.

   LogicPro returns you to the **Compiler** dialog box, if there are no errors.
3. Click **[OK].** This will launch the simulation.

NOTE! ***If an error occurred when you compiled LogicPro still returns you to the Compiler dialog box, but* [OK] *will not be enabled. You must click cancel and fix your program before proceeding.***

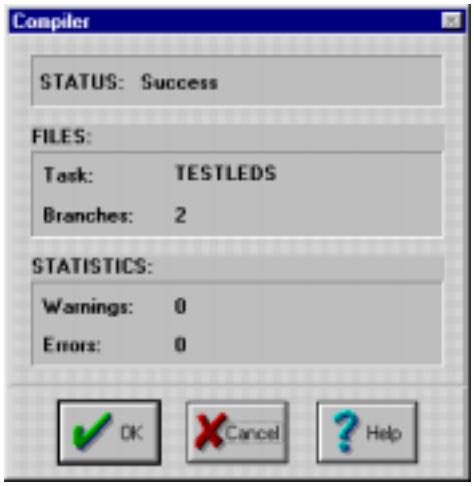


Figure 14.1 Compiler Dialog Box

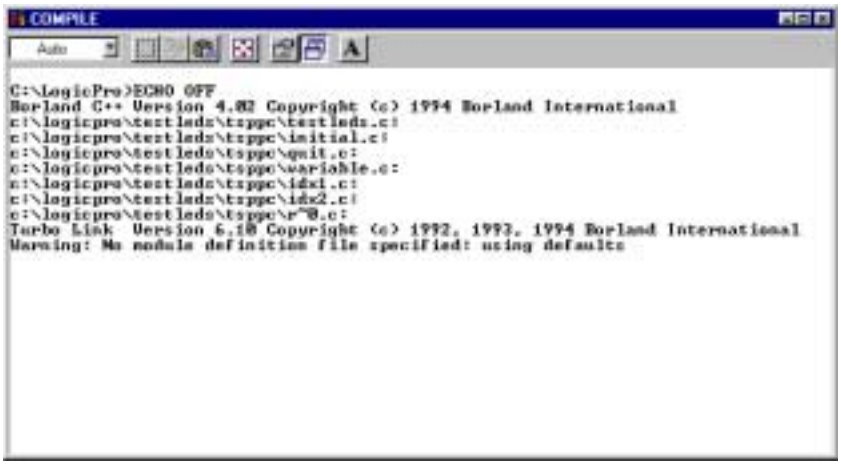


Figure 14.2 Borland C++ Compiler DOS window

Using the Simulation Tools

When in the Simulation mode, *LogicPro* will enable the Simulation tools on the Standard toolbar. Normally, these buttons are grayed out and unavailable. You must be in a Simulation before these buttons are available to you. *Table 14.1 on page 228* lists all of the buttons and their corresponding functions.

Table 14.1 Simulation Tools

This Button...	Performs This Function...
	Simulation Setup
	Increment Simulation Timer Ticks
	Scan Simulation Logic
	Simulation Reset
	Change Simulation Variables

Simulation Setup

To set up timers and scan rates:

1. Select the **Simulation** menu.
2. Select **Simulation Setup**
   – or –
   Click the **[Simulation Setup]** tool. This will cause the **Simulation Setup** dialog box to open, as shown in *Figure 14.3 on page 229*.
3. Click on the desired **Timer Tick** setting, the default setting is 1.
4. Click on the desired **Number of Scan** setting, the default is continuous.
5. Click **[OK]**.

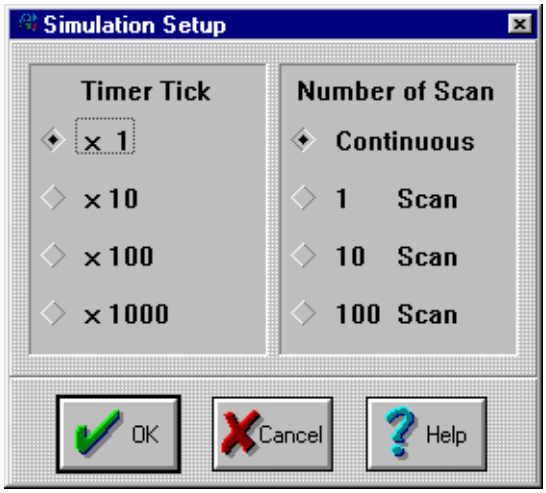


Figure 14.3 Simulation Setup Dialog Box

In the continuous scan mode, your program is free running but not in real time. The Timer Tick increases by the value you choose in the simulation setup. This allows you to multiply how much your program increments in time for each scan. The **Timer Tick** setting has the same effect on your simulation, regardless of the scan mode selected.

Timer Ticks

Timer Ticks set the amount by which timers are incremented on each scan of the logic.

To modify the Timer Ticks:

1. Select the **Simulation** menu.
2. Select **Simulation Setup...**
   – or –
   Click the **[Simulation Setup]** tool. This will cause the **Simulation Setup** dialog box to open, as shown in *Figure 14.4 on page 231*.
3. Click on the desired **Timer Tick** setting, the default setting is 1.
4. Click **[OK]**.

Increment Simulation Timer Ticks

Clicking the **[Increment Simulation Timer Ticks]** button allows you to multiply the active Timer Ticks setting by a factor of (n+1) where n = the number of times you click the button. This only effects the next Scan cycle.

For example, consider a Timer On Function Block. If you have the Timer Ticks set at 10 and click the **[Increment Simulation Timer Ticks]** button twice (n=2) then, in the next Scan cycle the timer will increment by 30 ticks.

Timer Tick Setting (10) x Number of clicks on the [Increment **Simulation Timer Ticks]** button (2 + 1) = (10) x (2+1) = 30.

Scan Simulation Logic

Clicking the **[Scan Simulation Logic]** button causes *LogicPro* to execute the number of scans selected in the simulation setup. For instance, if you selected 10 as the number of Scans, then pushing this button will result in an evaluation of all logic conditions 10 times.

Number of Scan

Modifying Number of Scan allows you to define how many times your program executes each time you click on **[Scan]**.

1. Select the **Simulation** menu.
2. Select **Simulation Setup**
   – or –
   Click the **[Simulation Setup]** tool. This will cause the **Simulation Setup** dialog box to open, as shown in.
3. Click on the desired **Number of Scan** setting, the default is continuous.
4. Click **[OK]**.

Resetting the Simulation

During simulation, you may wish to test several scenarios. *LogicPro* lets you reset the simulation and start over. Recompiling the simulation code is unnecessary.

By resetting the simulation all variables and constants will return to their original values. However, the simulation will retain all of the set up characteristics you have already created.

To reset the simulation:

1. Select the **Simulation** menu.
2. Select **Simulation Reset**
   – or –
   Click the **[Simulation Reset]** tool. This will cause the **Simulation Setup** dialog box to open, as shown in.

Change Simulation Variables

You can assign specific values to Variables to test the program under a variety of conditions. Select **Change Simulation Variable** from the **Simulation** menu. The **Change Variables** dialog box appears. Alternatively, you can push the **[Change Simulation Variables]** button from the Standard toolbar.

Once in the dialog box, you can assign or modify Variable values as often as you like. You can minimize or move this dialog box so as not to clutter your work place. This dialog box will automatically close when you close all of the simulation windows for the program or click **[OK]**.

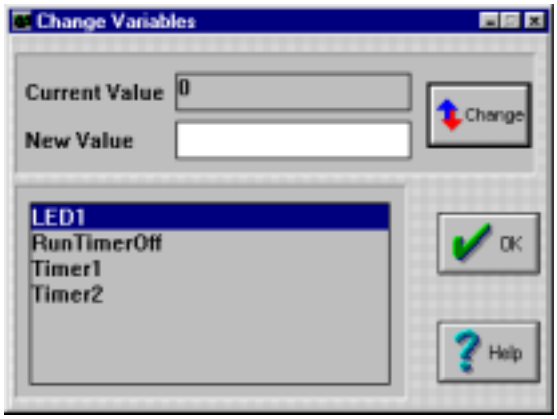


Figure 14.4 Change Variables Dialog Box

Variables whose values are determined by the program will not retain values set in the **Change Variables** dialog box. Logic scans will overwrite these variables' values.

NOTE! ***If you are not in the continuous scan mode, the logic will not reflect the changes to a variable until a scan takes place. Therefore, to see the effects you must trigger at least one scan cycle.***

Change Simulation Constants

In a Ladder diagram, double click on a function block to view and change a constant, see *Figure 14.5*.

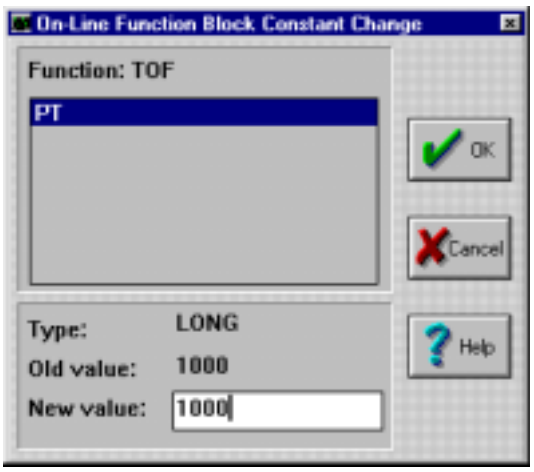


Figure 14.5 On-Line Function Block Constant Change

NOTE! ***Constants can only be changed in Ladder diagrams. You cannot change constants in a Function Block diagram.***

Viewing Individual Steps and Actions within a Simulation

In an SFC diagram, double-clicking on the Action to the right of the step opens a window displaying this Action in simulation mode.

This works regardless of the programming language used in the Action. If the Action is another SFC, you can continue to drill down and see more detail. Each time you double-click on the Action another window opens.

If the Action is a Ladder Diagram or a Function Block Diagram you can not go any deeper. However, you can change constants by double clicking on the Function Block. See the section entitled *Change Simulation Constants on page 232* for more information.

Closing the Simulation

1. Select the **File** menu.
2. Select **Close**.
3. Select **Simulation**.

15

Downloading and Monitoring

Executable

LogicPro converts your control logic program into a binary executable program. This program is in the form of a DOS ".exe". This file is converted to a hex file and is then downloaded to the controller and run.

Downloading to a controller through an RS-232 Port

To download the executable program generated by *LogicPro*, follow this simple procedure.

1. If downloading to a PPC-2000 controller:
   a. Set the control mode for each channel to **Off**.
   b. Set each alarm to **Disabled**.
2. If downloading to a CPC400 controller, note any alarms. Once downloading commences, all alarms are automatically acknowledged.

CAUTION! ***Input readings during downloads are not accurate and may vary widely due to allocation of CPU resources for downloading. Stopping closed loop control and disabling alarms ensures that outputs remain off during download.***

3. Connect the computer to the controller by a standard RS-232C connection through the COM1 or COM2 serial ports.

NOTE! ***The port is identified when setting up a resource. The port must be set correctly before building and compiling executable code for a resource. See Editing a Resource on page 36 in order to verify or change the communications port selection.***

4. Select **Execution menu.**
5. Select **Build All and Run**
   – or –
   Click on the **[Build All and Run]** tool. This will build the runtime executable, download it to the target controller and run your logic control program.
6. The **Downloading** dialog box will open during the download displaying transfer statistics and any errors encountered.
7. You can start and stop the run time executable by clicking the **[Start Executable]** tool
   – or –
   [Terminate] tool.

Alternatively, if a runtime executable has already been built:

1. If downloading to a PPC-2000 controller:
   a. Set the control mode for each channel to **Off**.
   b. Set each alarm to **Disabled**.
2. If downloading to a CPC400 controller, note any alarms. Once downloading commences, all alarms are automatically acknowledged.

CAUTION! ***Input readings during downloads are not accurate and may vary widely due to allocation of CPU resources for downloading. Stopping closed loop control and disabling alarms ensures that outputs remain off during download.***

3. Select the **Execution** menu.
4. Select **Download and Run**
   – or –
   Click the **[Download and Run]** tool on the Standard toolbar to just download the runtime executable to the controller and run your logic control program.

5. You can start and stop the runtime executable by clicking the **[Start Executable]** tool
   – or –
   [Terminate] tool.

6. The **Preparing for downloading** box will open during the preparation period of the download. Next, the **Downloading** box will display. See *Figure 15.1*.

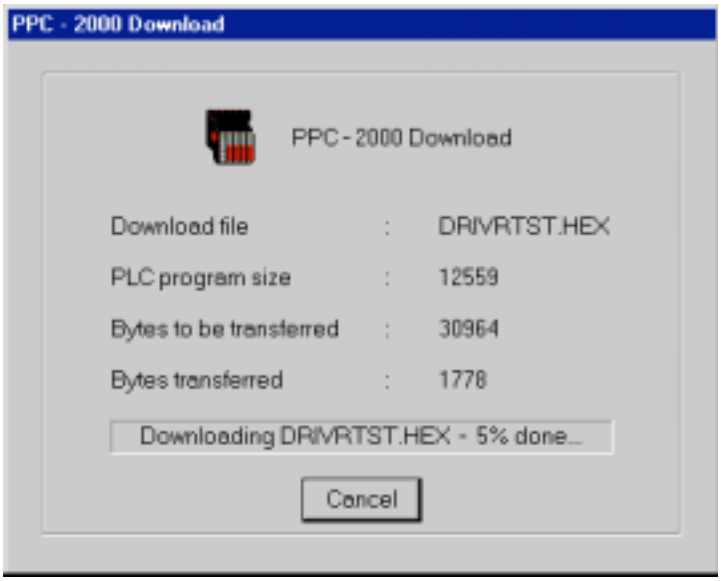

PPC - 2000 Download
PPC - 2000 Download
Download file : DRIVRTST.HEX
PLC program size : 12559
Bytes to be transferred : 30964
Bytes transferred : 1778
Downloading DRIVRTST.HEX - 5% done...
Cancel

Figure 15.1 Downloading

Monitoring a Program

To monitor a Program:

You can verify that the control logic is working properly by inspecting that status of different logic elements in a monitoring window. Program monitoring exchanges messages between *LogicPro* and the target resource. Note also that the status of the logic elements displayed on a monitoring window is NOT real-time.

1. Select the **View** menu.

2. Select **Monitor On/Off**
   – or –
   Click on the **[Monitor On/Off]** tool.

NOTE! ***Before you can use the monitor function you need to compile and download your program to a resource.***

3. *LogicPro* displays the most recently downloaded program window.

The flashing green dot on the right side of the Status-Bar indicates that *LogicPro* is actively communicating with the target resource. If you have more than one monitoring window open at a time, this flashing green dot indicates there are message exchanges between any one monitoring window and the target resource, not necessary the current active window. The individual window's title indicates when the last reply was received.

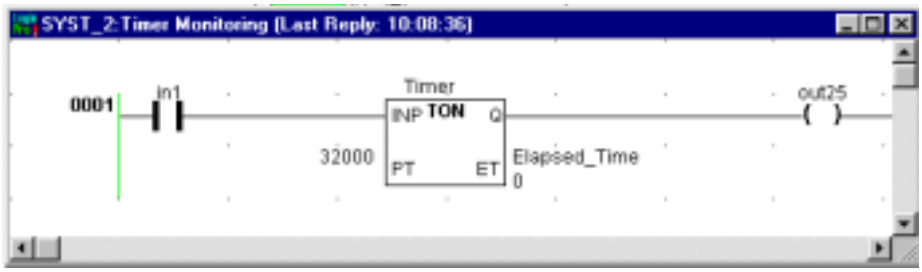


Figure 15.2 LogicPro Monitoring Window

To turn monitoring off:

You can temporarily disable the messaging between a particular monitoring window and the target resource by the following procedure.

1. Select the **View** menu.
2. Make the monitoring window become the active window.
3. Select **Monitor On/Off**
   – or –
   Click on the **[Monitor On/Off]** tool.
4. Repeat the above step to turn a monitoring window back on.

A check mark in front of the menu or a green block in the middle of the Speed-Bar button indicates the monitor is on for the current active monitor window.

NOTE! ***Having too many monitoring windows open at the same time can cause performance degradation. If this happens, close some of the monitoring windows or turn Monitoring Off for some of these windows.***

Monitoring Sequential Function Charts

LogicPro identifies the active step in the running SFC by highlighting it in green, see *Figure 15.3*.

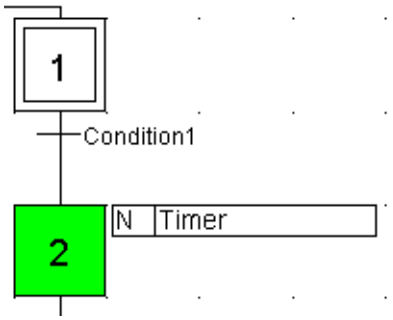


Figure 15.3 Active Step in an SFC

You can also monitor individual actions in Sequential Function Chart programs.

To open a monitoring window for an action:

1. Select the **View** menu.
2. Click on the action in the program.
3. Select **Monitor On/Off**
   – or –
   Click on the **[Monitor On/Off]** tool to toggle Monitoring on or off in each window.
4. When you are finished monitoring the SFC, select **Windows** then **Close All** to close the monitoring windows.

Monitoring Ladder Diagrams

In a Ladder Diagram, *LogicPro* indicates the state of a variable associated with an element by coloring it bright green when it is equal to one. This is true for all contacts and coils except the normally closed contact which is bright green when the variable associated with it is equal to zero. See *Figure 15.4 on page 240*.

Function blocks with variables assigned to Inputs or Outputs show the variables' current values.

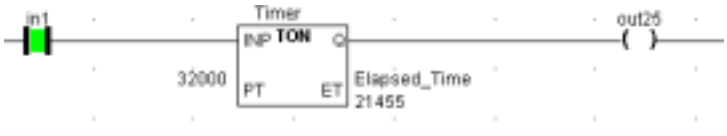


Figure 15.4 Monitoring Ladder Diagrams

Monitoring Function Block Diagrams

In a Function Block Diagram, only the variables change color to indicate activity. When a Boolean element is true (has a value of 1) that element is highlighted in green. Otherwise, monitoring an FBD program is the same as monitoring a Ladder program.

Values only appear with their associated variable block. If a variable is not associated with a variable block, its value will appear next to the Function Block.

NOTE! ***No online changes can be made in a FBD program.***

Forcing I/O

Boolean variables can be forced on or off while monitoring an SFC or ladder diagram program running in a controller.

CAUTION! ***Variables remain in their forced states until you clear or disable the forced condition(s), or terminate the logic program. Closing the monitor window or LogicPro will not clear or disable forced variables. Force variables only when you are sure it is safe to do so.***

To control I/O forcing while monitoring:

1. Right click on the **[I/O]** variable. The **Force-Clear I/O** dialog box appears.
2. Click one of the following buttons, depending on what you want to do:

- **[Force On]**. The value of 1 is given to the variable, forcing it on. A green square appears around the object.
- **[Force Off]**. The value of 0 is given to the variable, forcing it off. A red square appears around the object.
- **[Clear Force]**. Any forcing that the variable may have is removed.

3. Click **[OK]**. The changes take effect accordingly.

NOTE! ***Forcing is not supported in function block diagrams.***

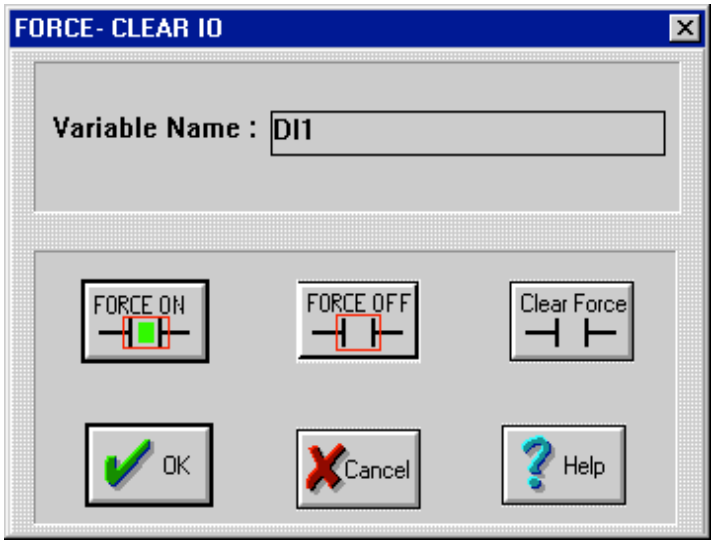


Figure 15.5 Force - Clear IO Dialog Box

Variable Watch

Variable Watch lets you easily view the current status and properties of program variables in a spreadsheet format. To use this feature you must:

- Configure the **Variable Watch** utility.
- Select variables in the **Watch Grid** (spreadsheet).

After configuring Variable Watch and selecting variables to view you can:

- View the values of the variables.
- Change the value of a variable as your program runs.
- Force Boolean variables.
- Save the current view and value settings.

Configuring Variable Watch

Variable Watch gets its data from FastLink, a separate communications program. Use *LogicPro* to configure FastLink.

1. If you have made changes or switched communications ports on the PC since you last compiled and downloaded the program, select **Build All and Run** from the **Execution** menu.
2. Select **FastLink Config** from *LogicPro's* **Option** menu. The **FastLink Configuration** dialog box appears.
3. Enter the `.FLK` file that corresponds with the project that you are working in.

NOTE! ***The name of the .FLK file is the same as the name of the project and resides in the same directory. The .FLK is created when you compile the project.***

4. Enter the **Access Time**.
5. Enter the **Message Timeout** time.
6. Enter the **Poll Rate**.
7. Enter the amount of retries.
8. Click **[OK]**.
9. Save your program(s) if necessary and close *LogicPro*.
10. To start FastLink, click **Start** on the **Windows** task bar, then click **Programs**, then **Watlow**, then **LogicPro CPC400/PPC-2000**, then click **FastLink**. FastLink runs in the background and appears on the task bar.

Selecting and Viewing Variables

To Select and View Variables:

1. To start **Variable Watch** click **Start** on the **Windows** task bar, then click **Programs**, then **Watlow**, then **LogicPro CPC400/PPC-2000**, then click **Variable Watch Window**.

NOTE! ***LogicPro must be closed and FastLink must be running for Variable Watch to work correctly.***

2. Choose **Select Grid** from the **View** menu.
3. Select the check box next to each variable whose properties and status that you want to view. A small red check mark appears in each selected box.
4. Select **Watch Grid** from the **View** menu. The **Variable Watch Window** displays the values and other properties of variables that you selected.

Changing the Value of a Variable

To change the value of a variable:

1. Double-click the **variable value** in the **Value** column.
2. Enter the new variable value.
3. Click **[Yes]**.

Forcing Variables

With the Variable Watch Forcing I/O feature:

- You can force a Boolean variable on or off, or clear the forced status for the variable.
- You can disable, re-enable or clear all I/O forcing in a program.

Forcing a Variable

CAUTION! ***Variables remain in their forced states until you clear or disable the forced condition(s), or terminate the logic program. Closing Variable Watch will not clear or disable forced variables. Force variables only when you are sure it is safe to do so.***

To force a variable on or off:

1. Start **FastLink**. (Click **Start** on the **Windows** task bar, then click **Programs**, then **Watlow**, then **LogicPro CPC400/PPC-2000**, then click **FastLink**.)
2. Start **Variable Watch**. (Click **Start** on the **Windows** task bar, then click **Programs**, then **Watlow**, then **LogicPro CPC400/PPC-2000**, then click **Variable Watch Window**.
3. Choose **Select Grid** from the **View** menu.

4. Select the check box in the **Select** column next to the variables you want to monitor in the Watch Grid.
5. Select **Watch Grid** from the **View** menu.
6. In the Watch Grid, right click on a variable.
7. Select **Force ON** to force a variable on, **Force OFF** to force a variable off, or **Clear Force** to clear the forcing of the variable.

Enabling, Disabling and Clearing Forcing

CAUTION! ***Variables remain in their forced states until you clear or disable the forced condition(s), or terminate the logic program. Closing Variable Watch will not clear or disable forced variables. Force variables only when you are sure it is safe to do so.***

To enable, disable, or clear all forcing:

1. Select **Forcing Options** from the **Options** menu.
2. Select **Enable**, **Disable**, or **Clear**.

Saving View Settings

To save view settings:

1. Click **File**, and then click **Save**. The **Save** dialog box appears.
2. Enter the name of the .PVW file.
3. Click **[Save]**.

Watch Grid View

The Watch Grid contains information about the variables you have chosen in the Select Grid. See *Figure 15.6 on page 245*.

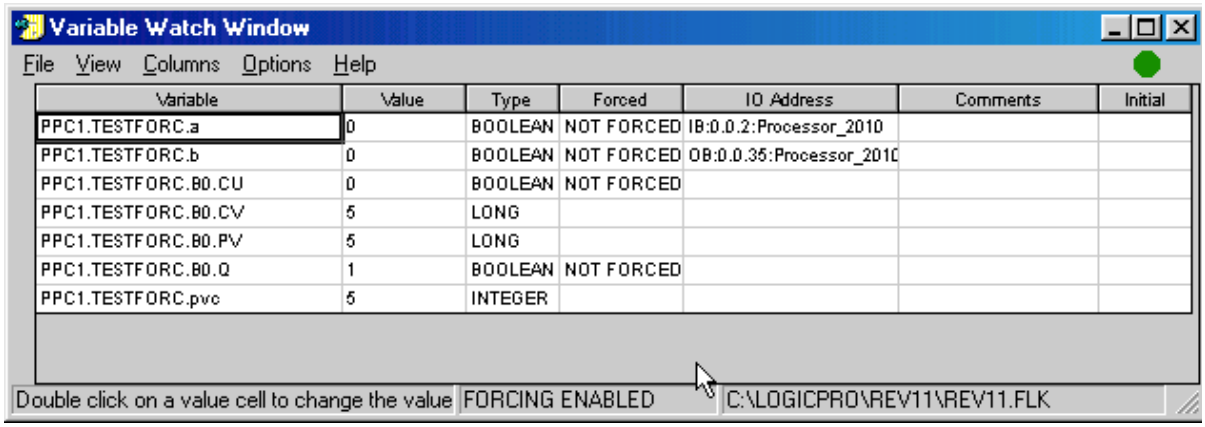

Variable Watch Window
File View Columns Options Help
Variable Value Type Forced IO Address Comments Initial
PPC1.TESTFORC.a 0 BOOLEAN NOT FORCED IB:0.0.2:Processor_2010
PPC1.TESTFORC.b 0 BOOLEAN NOT FORCED OB:0.0.35:Processor_201C
PPC1.TESTFORC.B0.CU 0 BOOLEAN NOT FORCED
PPC1.TESTFORC.B0.CV 5 LONG
PPC1.TESTFORC.B0.PV 5 LONG
PPC1.TESTFORC.B0.Q 1 BOOLEAN NOT FORCED
PPC1.TESTFORC.pvc 5 INTEGER
Double click on a value cell to change the value
FORCING ENABLED
C:\LOGICPRO\REV11\REV11.FLK

Figure 15.6 The Watch Grid in the Variable Watch Window

Table 15.1 The Columns in the Watch Grid

Column	Description
Variable	The name of the variable. In Variable Watch, the variable name contains the resource name, program name, and name that you gave the variable in your program – each separated by a dot. For example, for the variable SORTER.SIZER.Always_True: SORTER is the resource. SIZER is the program. Always_True is the variable
Value	The current value of the variable
Type	TypeThe type of variable, either Boolean, integer, long, or real. **NOTE:** You can only force Boolean variables

Column	Description
I/O Address	Input/Output address of the variable. In Variable Watch, the variable I/O address contains the following elements: • Type of variable: • I for input • Q for output • Size: X for bit B for byte W for word • IO Physical Address • Driver name For example, an I/O address in Variable Watch could be: IQ.X.0.0.2:Processor_2010.
Comments	Any comments that you attached to the variable when you created or last edited it.
Initial	The initial value that you gave the variable when you created or last edited it.
Retained	Indicates whether the variable retains the current value when the program stops running. If the cell is blank, the variable retains the current value when the program stops running and applies that value when the program starts again. If NO is shown in the cell, the variable resets to the initial value when the program starts again.

On-Line Constant Changes

LogicPro allows you to change all of the constants assigned to a program while monitoring it in either Execution or Simulation modes. All constants assigned to a Ladder diagram Function Block's input may be modified.

To modify a constant assigned to a Function Block's input:

1. Double-click the Function Block on the monitoring window. The **On-Line Function Block Variables Change** dialog opens, as shown in *Figure 15.7 on page 247*.

2. Select the variable you want to modify.
3. Enter the new value in the **New Value** field.
4. Click **[OK]** to send the new values to the controller
   – or –
   [Cancel] to abort these changes. The program window now displays the new constant values.

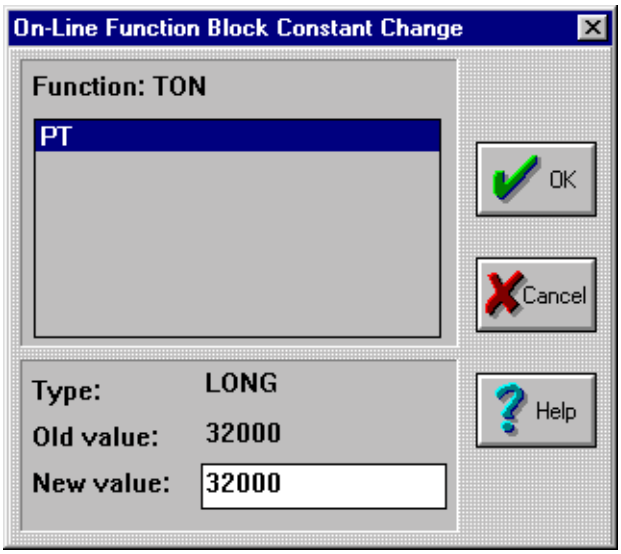


Figure 15.7 On-Line Function Block Variables Change Dialog Box

The change to the constant is made temporarily in the run-time program, not in the *LogicPro* program. The new constant value is valid while the program is running in the controller. If for any reason the controller's program is terminated and re-started, *LogicPro* restores the original constant values. The user will need to repeat the On-Line constant change procedure.

To keep the new constant value, it is strongly recommended that you update the constant value in the *LogicPro* program and download it again. This will make the new constant value permanent.

To update the constant value in the *LogicPro* program, follow these steps:

1. Open the program.
2. Make the change.
3. Save the program to disk.
4. Compile.
5. Download again.

16

Cross-Referencing

Cross-referencing helps you to debug your application Programs. You can also use cross-referencing to document Programs and track the locations of Variables and Program elements.

Cross-referencing is the equivalent of a search command for graphical control Programs. You can use both on-line and off-line Print Functions. You can customize printed output to show only the information you need.

Using Cross-Referencing

Cross-referencing is available in all three programming languages.

SFC On-line Cross-Reference

To perform an SFC on-line cross-reference, while in an SFC window:

1. Select the **Tools** menu.
2. Select **Cross Reference**
   - or -
   Click the **[Cross Reference]** button on the toolbar. The standard pointer changes into a modified pointer, as shown in *Figure 16.1 on page 250*.
3. Click on the step, transition, or action for which you want to view a cross-reference. The **Cross Reference** dialog box appears. This dialog lists all the Variables present for that step, transition, or action and its location in the Program.

4. You can minimize and move the **Cross Reference** dialog box so that you can view and edit portions of the Program. If you change the Program or Variables, the **Cross Reference** dialog box data is not updated until you close and then reopen the dialog box.
5. Click **[Print]** to print the on-line cross-reference listing.
6. Click **[OK]** to close the dialog box.

Figure 16.1 The Cross-Reference Pointer

Ladder On-line Cross-Reference

To perform a Ladder On-line cross-reference, while in a Ladder window:

1. Select the **Tools** menu.
2. Select **Cross Reference**
   - or -
   Click the **[Cross Reference]** button on the toolbar. The standard pointer changes into a modified pointer, as shown in *Figure 16.1*.
3. Click on a Ladder element, Function Block, or Rung for which you want to view a cross-reference. The **Cross Reference** dialog box appears. This dialog lists all the Variables present for that element, Function Block, or Rung and its location in the Program.
4. If you click outside all elements, the **Cross Reference** dialog box appears and lists all the Variables present for the entire ladder window and their location in the Program.
5. You can minimize and move the **Cross Reference** dialog box so that you can view and edit portions of the Program. If you change the Program or Variables, the **Cross Reference** dialog box data is not updated until you close and then reopen the dialog box.
6. Click **[Print]** to print the on-line cross-reference listing.
7. Click **[OK]** to close the dialog box.

Function Block Diagram On-line Cross-Reference

To perform a FBD On-line cross-reference, while in an FBD window:

1. Select the ***Tools*** menu.
2. Select **Cross Reference**
   - or -
   Click the **[Cross Reference]** button on the toolbar. The standard pointer changes into a modified pointer, as shown in *Figure 16.1 on page 250*.
3. Click on a Variable for which you want to view a cross-reference. The **Cross Reference** dialog box appears. This dialog lists all the instances of that Variable and the location of each instance.
4. If you click outside all elements, the **Cross Reference** dialog box appears and lists all the Variables present for the entire FBD window and their location in the Program.
5. You can minimize and move the **Cross Reference** dialog box so that you can view and edit portions of the Program. If you change the Program or Variables, the **Cross Reference** dialog box data is not updated until you close and then reopen the dialog box.
6. Click **[Print]** to print the on-line cross-reference listing.
7. Click **[OK]** to close the dialog box.

Cross-Reference Output

The cross-reference output is sorted by type in the following order:

1. Project Variables
2. Resource Variables
3. Program Variables

Within each of these categories, the Variables are sorted in alphabetical order. If Comments for the Variable exist, they are displayed on the line below the Variable Code. The last information displayed is the locations of the Variables.

Each SFC and Ladder Rung is listed on a separate line. The items on each of the lines are:

- The name of the Program in which this Variable is used.
- The action in which this Variable is used. There may be more than one action on a line.
- The rung number in which the Variable is located.
- For Ladder Rungs, LogicPro displays either the type of Element or name of the Function Block for each Variable used.

LogicPro displays function block names between two dashes (- -).

The following text is an example of a Variable location:

```
Program Variables
Var1:BOOL;
Program
Program:Action:Ladder:0 -| |- -( )-
Var2:INT;
Program:Action:Ladder:0 -| |- -
```

This example shows the cross-reference output of the Program Variables. This Program uses two Program Variables; Var1 and Var2.

Output Printing

You can print the on-line cross-reference display by selecting **[Print]** in the **Cross Reference Display** dialog box. This method of printing is discussed in *Using Cross-Referencing on page 249*.

You can also print Program documentation that includes the cross-reference.

To print Program documentation including the cross-reference:

1. Select the **File** menu.
2. Select **Print**.
3. The **Print** dialog box appears, as shown in *Figure 16.2*.
4. Select to output documentation about:
   - The entire Program
     -or-
   - The current window
     -or-
   - The current window and all actions below it.
5. Click **[Print Setup]** to define which printer you want to use to print the output documentation.
6. Click **[Report Setup]** to define what information is included in the report. For more information on defining the report's components, see *Report Setup on page 254*.
7. Click **[Print]** to send the output to the printer.

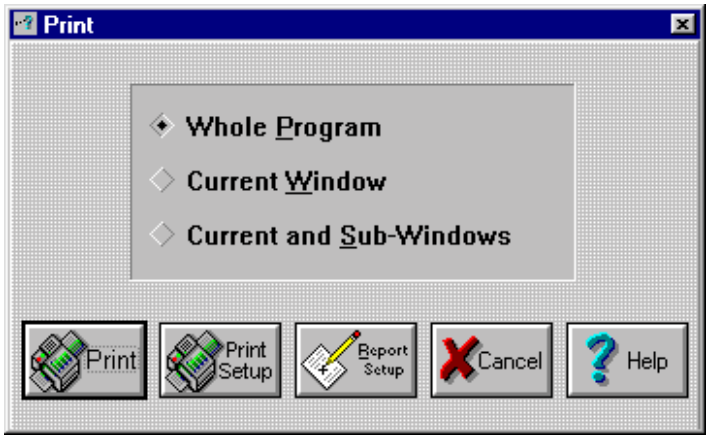


Figure 16.2 Print Dialog Box

Report Setup

In the **Report Setup** dialog box, as shown in *Figure 16.3 on page 254*, you can select:

- What Variable information is printed in the cross-reference Variable codes, by using **Variables Setup**.
- Ladder documentation, by selecting **Ladder**.
- Specific information included in the Ladder documentation output, by clicking the **Ladder Setup** button.
- SFC documentation, by selecting **SFC**.
- Specific information included in the SFC documentation output, by clicking the **SFC Setup** button.
- FBD documentation, by selecting **FBD**.
- Specific information included in the FBD documentation output, by clicking the **FBD Setup** button.
- Cross-reference for the entire Program, by selecting **Cross Reference** .
- I/O Table, by selecting **I/O Table**.

The I/O Table is a list of all Program Variables that use I/O Drivers, sorted by I/O Driver.

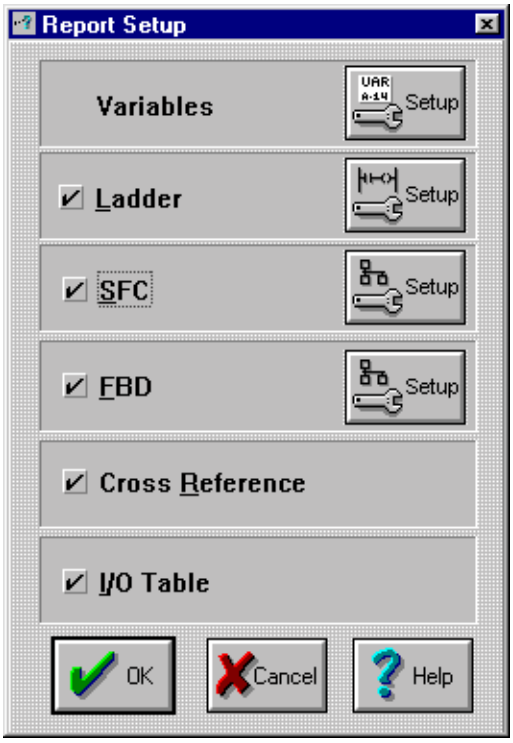


Figure 16.3 Report Setup Dialog Box

Variables Setup

In the **Variables Setup** dialog box, as shown in *Figure 16.4 on page 255*, check the boxes corresponding to the information to be included in the Variable Report:

- If comments about the Variable are included on the line after the Variable code, by selecting **Comment**.
- If the Initial Value for the Variable is included, by selecting **Initial Value**.
- If the I/O Address for the Variable is included, by selecting **I/O Address**.
- If the I/O Driver for the Variable is included, by selecting **I/O Driver**.
- If the Size for the Variable is included, by selecting **Size**.
- If the Type for the Variable is included, by selecting **Type**.

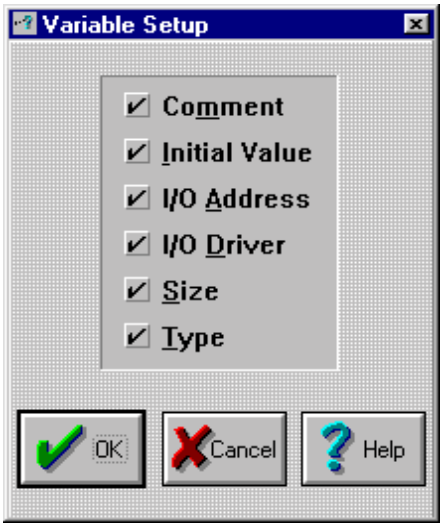


Figure 16.4 Variable Setup Dialog Box

Ladder Setup

In the **Ladder Setup** dialog box, as shown in *Figure 16.5 on page 256*, you can select what information is included for each Rung of a Ladder. For example, you can select:

- If the rung graphic is output, by selecting **Graphic**.
- If the rung comments are output, selecting **Comments**.
- If a cross-reference of the Variables in the rung are output, by selecting **Cross Reference**.

- If **Cross Reference** is checked, choose whether the **All Elements** or only **Coils** are included in the cross-reference output.

LogicPro automatically resizes Ladder diagrams to fit the page. This results in smaller images.

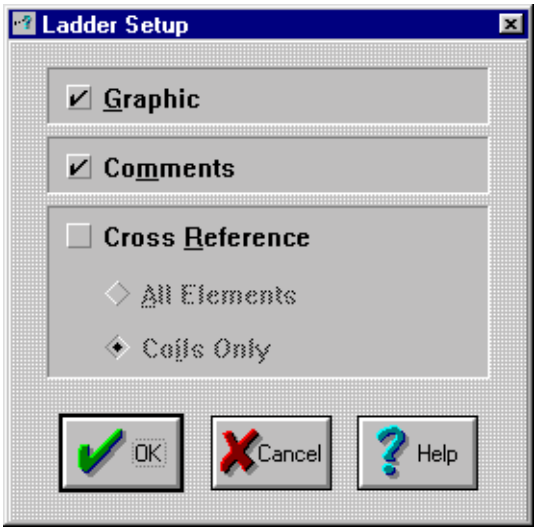


Figure 16.5 Ladder Setup Dialog Box

SFC Setup

In the **SFC Setup** dialog box, as shown in *Figure 16.6*, you can select what information is included in reports for each SFC. For example, you can specify:

- If the SFC graphic is output, by selecting **Graphic**.
- If a cross-reference of the Variables in the SFC is output, by selecting **Cross Reference**.

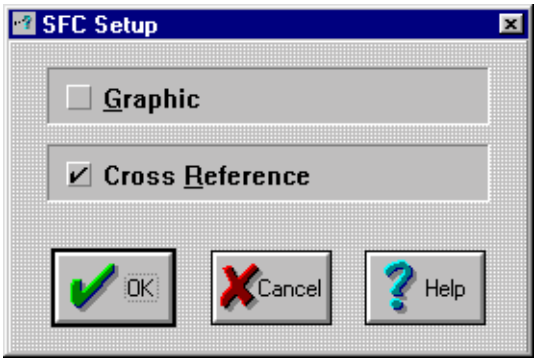


Figure 16.6 SFC Setup Dialog Box

FBD Setup

In the **FBD Setup** dialog box, as shown in *Figure 16.7*, you can select what information is included for each SFC. For example, you can specify:

- If the FBD graphic is output, by selecting **Graphic**.
- If a cross-reference of the Variables in the FBD is output, by selecting **Cross Reference**.

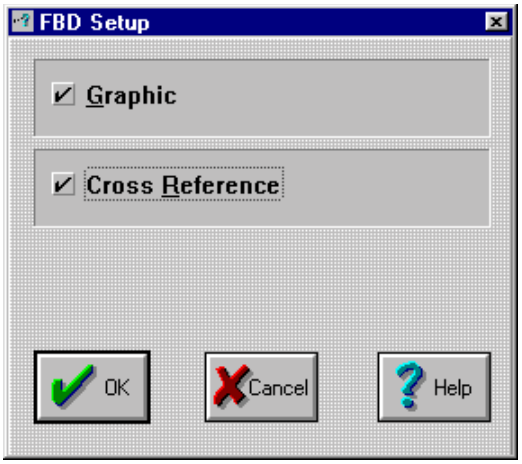


Figure 16.7 FBD Setup Dialog Box

17

Dynamic Data Exchange

Dynamic Data Exchange (DDE) is a communication protocol designed to allow Windows applications to send and receive data and instructions to other Windows applications.

Dynamic Data Exchange takes place between a client application and a server application. The client initiates the exchange by establishing a conversation with the server so that it can send transactions (requests for data or service) to the server. The server responds by providing the requested data or services.

A server can have many clients at the same time, and a client can request data from multiple servers. Applications can be both client and server. The client terminates a conversation when it no longer needs the server's data or services.

How to Access a Remote Item

A DDE Server uses a three-level name to identify the data unit that the server can exchange:

- **Server name** — is a string that a server application responds to when a client attempts to establish a conversation with a server. A client must specify this service name to be able to establish the conversation. Although a server may respond to many server names, most servers respond to only one. In the case of *LogicPro*, the server name is LP_DDE.

- **Topic name** — is a string that identifies a logical data context. For LogicPro, this is a unique name that identifies the resource. The actual resource is defined in the `.DDE` file. The client application must specify a topic name along with a service name when it attempts to establish a conversation with a server.
- **Item name** — is a string that identifies a logical data context. Items can be any pre-defined symbolic Variable name. The Variable name in the user Program is used by *LogicPro* as the item name.

For the client, these names are key for establishing a conversation and receiving data from a server.

Installation

The *LogicPro* DDE Server is automatically installed at the same time *LogicPro* is installed.

Starting the LogicPro DDE Application

To start the *LogicPro* DDE application:

1. Click the start button on the Windows task bar.
2. Select **Programs**.
3. Select **Logicpro**.
4. Select **DDE Server**.

Configuration

Before you can open a topic, you must configure the *LogicPro* DDE Server. The major steps in this process are:

- Create *LogicPro* DDE Server
- Open the Server
- Configure the Server

To configure the *LogicPro* DDE Server:

1. Select the **Configuration** menu
2. Select **Configuration**. The **DDE Configuration** dialog box appears.

3. Enter a time in milliseconds in the Update Interval field. This field sets up an internal timer that acts as the time base for polling the controller for Program status.
4. For example, if the Update Interval is set at 3 milliseconds, the LogicPro DDE Server polls the controller and reads the data every 3 milliseconds.
5. Check the **Load Default Topic on Start-up** box.
6. Click **[Setup]**. The **Select a Topic as the Default Topic** dialog box appears.
7. Select a topic from the DDE topic list.
8. Select the **Don't show warning messages** option.
9. Click **[OK]** to close this dialog, and then click **[OK]** to close the **Logicpro DDE Configuration** dialog box.

All parameters set in the Configuration menu are saved in the ASCII text file `LP_DDE.CFG`. You can view this file at any time.

CAUTION! ***Never modify the*** `LP_DDE.CFG` ***file directly with a text editor.***

Creating the Application FPR File

Before continuing, you need to create the Dynamic Data Exchange file to be used with the *LogicPro* DDE Server. An `.FPR` file is created and used by the *LogicPro* DDE Server during code generation of your Program. This filename is the same as the application and is placed in the resource directory.

This `.FPR` file contains all the information the *LogicPro* Server needs to establish communications with the controller. The file information is updated every time code is generated for your Program. This `.DDE` file contains the following information:

- **Program ID** — a unique number that matches this file to the executable file `(.EXE)` in the controller with the `.FPR` file.
- **Target —** the controller resource you are using.
- **Port** — how and where the controller is attached, such as through Com1 serial port.
- **Variables** — a list of all symbolically named Variables in the control Program. The list contains:

- The symbolic name.
- Type of Variable (`BOOL=0, INT=1, REAL=2`).
- A Program index.

The Variable names are the same as the item names used by other application Programs to make requests through the DDE Server.

NOTE! ***Do not modify this file. LogicPro will make any corrections as they are needed.***

For more detailed information, refer to *Chapter 6, Variables.*

Topics

The following commands for topics are accessed through the **Topic** menu:

- **New Topic**
- **Open Topic**
- **Close Topic**
- **Modify Topic**
- **Delete Topic**

Creating A New Topic

Before any communication can begin between the *LogicPro* DDE Server and client applications, you need to create a topic.

To create a New topic:

1. Select the **Topic** menu.
2. Select **New Topic**. The **New DDE Topic** dialog box appears.
3. Enter a topic name in the **Topic** field.
4. Enter the name of the `.FPR` file. If you are not sure of the file's name or location, click **[Search]** to open a standard Windows search dialog to locate the file.
5. Click **[OK]**.

Open a Topic

To Open a topic:

1. Select the **Topic** menu.
2. Select **Open Topic**. The **Select DDE Topic** dialog box appears.
3. Choose the topic you want to use from the list.
4. Click **[OK]**. Only one topic can be open at a time.

Once the topic is selected, the *LogicPro* DDE server is ready to run. At this point, the necessary configuration is read from the .FPR file and all hardware is initialized.

Close a Topic

To Close a topic:

1. Select the **Topic** menu.
2. Select **Close Topic**. The DDE Server closes the topic.

Modify a Topic

You can modify a topic by changing the .FPR file with which it is associated. As long as the topic is not open, it can be modified.

To Modify a topic:

1. Select the **Topic** menu.
2. Select **Modify Topic**. The **Select a Topic to Modify** dialog box appears.
3. Select the topic you want to modify.
4. Click **[Edit].**
5. Change the name of the .FPR file with which you are creating the client-server relationship.
6. Click **[OK],** then click **[OK]** again.

Delete a Topic

To Delete a topic:

1. **Select the Topic** menu.
2. Select **Delete Topic**. This removes the topic from the list of available topics.

NOTE! ***Once the topic is deleted, it cannot be recovered. Make sure you really want to delete the topic before you do so.***

Item Point Naming

In *LogicPro*, an item is a symbolically named Variable. Items can be read or written to through the *LogicPro* DDE server with the exception of real world I/O points. If the values of these points are changed by the *LogicPro* DDE Server, the scan loop of the controller will overwrite these changes.

Finding The Available Item Names

Once a topic is opened, you can acquire a list of available symbolic names for that application Program by selecting the **Topic Info** menu then **Variables**. The **Variables List** dialog box appears with a complete listing of available names, including Real-time values.

Excel Spreadsheet DDE Sample

In this example, Excel is used with the LogicPro DDE Server. To run this example, you need:

- LogicPro v2.0 or greater.
- LogicPro DDE Server.
- Microsoft Excel v4.0 or greater (v5.0 is strongly recommended).

The Program, Example 1, shown in *Figure 17.1 on page 265*, continuously increments six timers.

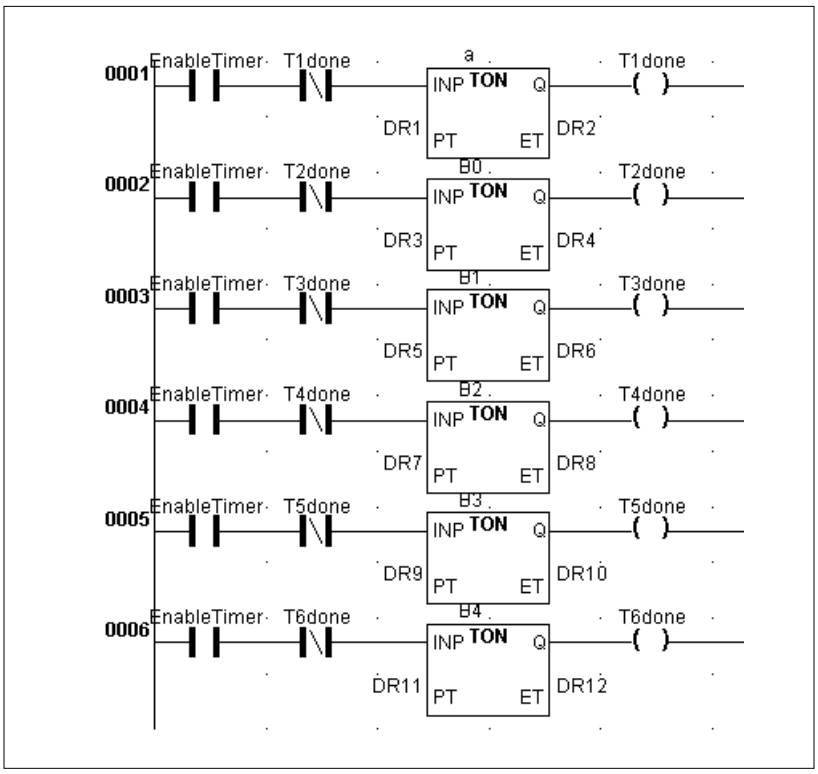


Figure 17.1 Example 1

NOTE! ***Before you can open LogicPro, you must close the DDE Server. Before downloading the Example1 Program, make sure that the resource is set to the correct platform.***

To download the Example 1 Program:

1. Open the **LogicPro Development Environment.**
2. Select the **Execution** menu.
3. Select **Build All and Run.**
4. *LogicPro* compiles the Program and automatically downloads it to the target platform.
5. To verify that the Program is successfully running, you can select the **View** menu and then **Monitor On/Off**. If the timer is counting, the Program is running and you can start the *LogicPro* DDE Server and begin the Excel spreadsheet example.

NOTE! ***You must exit LogicPro to free the communication channel or an error box will appear when the LogicPro DDE is started.***

Setting Up Excel to Read

Using the spreadsheet as a sample, the DDE Server reads the Variables and places their values in the specified cells. The spreadsheet reads Variables and shows a chart reflecting those values, as shown in *Figure 17.2*.

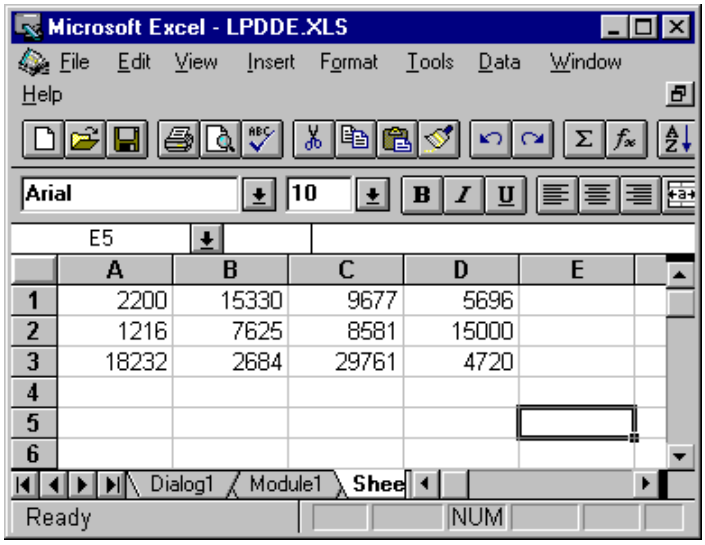


Figure 17.2 Spreadsheet Example

To specify a specific Variable inside the controller, call the `LP_DDE` Server and request the data. In the DDE Server, use Topic1 which you configured earlier. Topic1 uses the `.FPR` file for the configuration of the controller.

Excel provides an easy way to read a value from a DDE and place it into a cell. The correct format is:

```
=SERVER NAME|TOPIC NAME|ITEM NAME
```

- `Server Name`— `LP_DDE.` This is the executable file name for the LogicPro DDE server.
- `Topic Name` — the name that was assigned at the **New Topic** dialog box.
- `Item Name` — the symbolic Variable name that is used in the Program.

The A1 cell will contain the following string:

```
=LP_DDE|TOPIC1!DR1
```

NOTE! ***The topic and server names are delimited by a pipe (|). The topic and item name are delimited by an exclamation point (!). For more information on using Microsoft Excel and Dynamic Data Exchange, see your Microsoft Excel documentation.***

Glossary

A

Action

An operation performed in a control process as part of the SFC step.

Active Window

The window receiving active input from the mouse and keyboard.

Address

1) The physical location of data in a computers main memory or on a hard disk drive.

2) The unique name given to a component in a network.

API

Application Program Interface, a standardized set of tools, routines and protocols. Programs written for the Windows API make it easier for the end-user to learn because the Program will have the look and feel of any other Windows program.

Append

To add data to the end of an existing file without overwriting any of the existing data.

Application

Programs designed for end-users, as opposed to Programs that supports systems operations. Examples of applications include databases, word processors and *LogicPro*.

ASCII

American Standard Code for Information Interchange, a code for representing English character as numbers, with each letter assigned a number from 1-127.

ASCII Based Editor

See **Text Editor**.

ASCII File

A text file in which each byte is equivalent to a character as defined in ASCII. Sometimes referred to as *plain text files*.

Associate

Linking a data file with an application. In MS-DOS and Windows environments the three letter extension forms the basis for association. For instance the ".doc" extension identifies a file as a MS Word document.

Attributes

Characteristics of a file indicating several of its properties such as; read-only, archive, compressed, hidden and system.

B

Back Up

The practice of maintaining a duplicate copy of important data at an alternative location. Thereby guarding against data loss due to mechanical failure of a drive.

Baud Rate

The number of oscillations, or bits transmitted per second. A common unit of measure for data transmission.

BIOS

An acronym for Basic Input Output System. This is firmware that resides on your computer's motherboard. It provides the essential operating instructions allowing the CPU to address the monitor, keyboard and disk drive prior to loading an Operating System.

Bit

A binary digit, having a value of zero or one.

Bitwise

A logic tool that allows you to apply the principles of Boolean Logic to the individual bits of a byte. Not all programming languages support bitwise logic, but C does support a handful of operators:

>> (Shift bits left)

<< (Shift bits right)

& (AND - compares two groups of bits)

| (OR - compares two groups of bits)

^ (XOR - compares two groups of bits)

~ (complements a group of bits)

Block

In *LogicPro*, block may refer to either Function Blocks or block of data. A block of data represents input or output from *LogicPro* to a controller Resource. *LogicPro* can transfer data in blocks (e.g., to integer variables) between the IO and the PC.

Boolean

Variable condition that is either True or False and having a size of 1 bit.

Boolean Logic

A system of logic invented in the 1850's and named after its inventor George Boole. This is the study of operations that can be carried out on variables that can only have on of two values 0 (FALSE) and 1 (TRUE).

Borland C++

A programming language marketed by Borland and based on the C programming language.

Branch

On a Ladder Diagram, parallel spur used to create "OR" and other logic structures.

Build

In *LogicPro*, the act of compiling an SFC, Ladder or FBD Program. A build results in an executable file, i.e., *.exe or *.dll.

Button

Item on the toolbar (such as a step or function block) that can be selected for insertion into the process control Program. Toolbar buttons remain active until another button is selected.

Button, Radio

See **Radio Button**.

Byte

A sequence of eight bits. Range 0 - 255.

C

C

A small robust high-level programming language developed at Bell Labs in the 1970's by Brian Kernighan and Dennis Ritchie.

Cascade

Arrangement of open Program windows so just their title bars are displayed, with the current active Program window in full view.

Case Sensitive

The ability of a Program to differentiate between upper and lower case letters. Programs that can make this distinction are called "case sensitive" and would view Luck, luck, LUCK and LucK as four different entries.

Checkbox

Dialog box item that toggles a setting between On and Off. When the option is On, an X appears within the checkbox.

Circuit Element

The various graphic elements that make up a *LogicPro* Project such as; coil, contact etc.

Click

Depressing one of the buttons on your mouse or pointing device.

Click, Double

Click twice in rapid succession, usually with the left mouse button of your pointing device.

Click, Left

Depressing the left mouse button on your mouse or pointing device.

Click, Right

Depressing the right mouse button on your mouse or pointing device.

Clipboard

A Windows Application that captures and temporarily stores information. Used primarily in editing, this is the area where cut and copy information resides until replaced with new data or shutting down the system.

Cloning

A method of quickly duplicating variables within a Project.

Coil

Electrical element utilized to energize switch contacts, motors, etc.

Coil, Normal

A coil that enables an output when it is energized (a TRUE condition is present).

Coil, Unlatched

A coil that, when powered, resets to the off condition. Sometimes referred to as a "Reset Coil."

Coil, Latched

A coil that, when power is available from the left power rail, is set to the on condition and remains on until reset. Sometimes referred to as a "Set coil."

COM Channel

Serial communication path through which a PC transmits and receives data.

Compiler

Computer program (such as Borland C++) that generates machine language code that is eventually loaded onto the Resource controller.

COM Port

A computer interface used for connecting and communicating with external devices.

COM1

The name of a serial communication port. This is the unique name of one of several serial ports on a PC.

COM2

The name of a serial communication port. This is the unique name of one of several serial ports on a PC.

Configuration File

A file that contains operating parameters for an application or operating system. For example, in Windows "*.ini" files contain important Windows configuration information. While in the MS-DOS environment this information coexists with information for other Applications in the config.sys file.

Constants

A value used in programming that never changes during the lifetime of an application. The other type of value used in programming is the variable and its value may change during Program execution.

Contacts

Nodes through which energy flows, such as normally open and normally closed contacts.

Control Logic

The characteristic behavior of Programs created in *LogicPro* (i.e., SFC, Ladder and FBD). The commands issued by these Programs to govern the operations of machinery or other hardware.

Controller

The device that accepts and runs the logic Program created with *LogicPro*.

Cross Reference

LogicPro feature that enables the developer to view the settings of variables to help discover logic or Program errors.

Current Directory

This is the directory in which you are currently working. The operating system assumes that all referenced files exist in this directory unless you specify a different path.

D

Dialog box

On-screen box accessed to view and adjust settings and input information.

DDE

See **Dynamic Data Exchange**.

Debug

The art of finding and correcting errors in a design or Program.

Declaration

In programming, this is the act of assigning a name and data type to a variable or constant.

Default Value

The initial value of a variable if you do not declare an initial specific value.

Device

A component or machine that attaches to a computer. Examples of devices include printers, modems, controllers, etc. Each device requires software, called drivers, to allow the computer and the device to communicate with each other.

Directory

Part of an operating system's organizational hierarchy. The directory is a file that keeps information on other files that are, figuratively speaking, stored within it.

Directed Links

SFC element which, with transitions, directs the Program flow from one step to another.

DLL

An acronym for Dynamically Linked Library.

Documentation

Feature of *LogicPro* that allows the developer to attach text and comments to Program elements in the control process Program.

Download

The transfer of data, in the form of a file, from a source device to a peripheral device.

Downloading

The act of performing a download.

Drive

A shortened form of diskdrive. Any media on which data can be stored and retrieved.

Driver

Software that allows a computer and a device to communicate with each other. See **Device**.

Dynamic Data Exchange

Communication protocol that allows *LogicPro* to send and receive data and instructions to other Windows applications (such as Excel) in a client-server relationship.

E

Element

Basic programming unit of *LogicPro*, such as a step or transition (in SFC), or a coil or function block (in Ladder diagram).

Executable File

The output of compiled source code or object code which can be read and executed directly by a computer processor.

Executable Path

Operating System Path where the Executable File resides.

Execution Library

See **Library**.

Export Variables

Saving Variables from *LogicPro* Programs, Projects or Resources. These variables are normally written to a reusable file.

Expression

A combination of symbols, constants and variables that represent a value and are recognizable to the programming language.

F

FALSE

In Boolean Logic and Algebra FALSE is one of two possible values of a logical expression. TRUE is the other condition.

FBD

See **Functional Block Diagrams**.

Field

An area set aside for specific data. For example, an Employment Application will contain several fields for your name and your address. In database management you will find three types of fields: required, optional and calculated.

FIFO

First In First Out.

File Extension

In Microsoft Disc Operating System, this refers to the three characters following the "." in a file name. For example, "txt" is the file extension for the file named "readme.txt."

Flag

A hardware or software switch that can be either on or off and signals a specific condition is present.

Flag Variable

Generally, a Boolean variable used for identifying a condition or event in a Program.

Flash Disk Drive

An EEPROM that can be programmed in place by using specific algorithms.

Folder

See **Directory**.

Function

A part of a Program that performs a specific task.

Function block

A controller programming element consisting of a data structure partitioned into input, output, and internal variables.

G

Global Variable

Variable which can be used in all Resources within a Project or all Programs within a Resource.

GUI

Graphical User Interface

Graphic User Interface

A type of environment, making extensive use of a computers graphic abilities, which defines your interaction with the computer and its software. Windows

and Mac-OS are two popular examples of a Graphic User Interface (GUI).

Graphical Language

A programming language that makes extensive use of graphic symbols for easier programming. This allows you to draw a Program, instead of coding one. *LogicPro* supports Ladder, SFC and FBD Graphical Language.

Graphical Object

Symbols used to represent functions in a Graphical Language. In *LogicPro* examples of Graphical Objects include contacts, coils, blocks etc.

I

I/O

Input output this refers to any process that exchanges data through a computer, either writing data to it, retrieving data from it or both. Typically this term is used to identify software and hardware that are non-computational in nature. For example, a printer or a hard drive are I/O devices since they are non-computational but a CPU is not since it computational.

I/O Address

A unique identifier for I/O devices. Computer Program used by *LogicPro* to transfer information to and from a hardware controller Resource or other peripheral device.

I/O Driver

Computer Program used by *LogicPro* to transfer information to and from a hardware controller Resource or other peripheral device.

Icon

A visual symbol in the Windows environment.

IEC 1131

International Electrotechnical Commission standard designed to enhance interpretability by providing a common programming language across a variety of controllers and platforms.

IEC

The International Electrotechnical Commission.

Import Variables

Retrieving saved Variables from a reusable text file for *LogicPro* Programs, Projects or Resources.

Input

The action of reading values into a Program, process or hardware device.

Input Variable

Variables used for writing values into a Program, process or hardware device.

INT

See **Integer**.

Interface

The junction of two separate processes or devices. For example, to print a letter you must interface with an operating system to launch a word processor. Next you need to interface with the word processor, via your keyboard, to enter your letter. Next, the word processor needs to interface with the computer. Finally, your computer needs to interface with a printer to create the letter.

Interrupt

A signal sent to a Program to inform it that an event has occurred. Both hardware and software are capable of generating interrupts.

Integer

A positive or negative number with no fractional parts or zero.

L

Ladder Diagrams

An IEC 1131-3 standardized set of symbols and functions used to graphically design logic using relays. This diagram consists of left and right power rails connected by "rungs" on which relays and coils reside.

Language

A system for communicating instructions to a computer using a defined syntax. There are several major types of languages used by computers: machine, programming and fourth-generation are examples of a few of these language types.

Library

A collection of frequently used modules, precompiled routines in object code form.

LIFO

Last In First Out.

Link

A connection that exists between modules and routines as a result of the actions of a "linker." The result of this process is the creation of an executable file from these otherwise unconnected modules.

Logic

See **Boolean Algebra**.

LogicPro

IEC 1131 compliant Microsoft Windows-based, graphical programming environment allowing developers to create, debug, and manage Real-Time process control Programs written in traditional programmable controller languages.

Loop Back

Transition which passes process control from one step back to a preceding step.

M

MAX

SCAN Maximum Value Variable.

Menu Bar

Located near the top of the screen and displays the names of the pull down menus which can be accessed.

Message Box

A small box that appears on the screen to advise you of a problem or potentially dangerous operation.

MIN

SCAN Minimum Value Variable.

Modem

An acronym for Modulator - Demodulator. A device that allows you to interface your computer with a non-ISDN telephone line. Since conventional phone lines transmit analog data the MODEM must translate the computers digital data stream into analog data for transmission. When data returns to your computer from the phone line the MODEM translates the analog signal back into a digital signal.

Monitoring

A feature in *LogicPro* that allows you to observe the status of a Program while it executes.

MOVE

A function block that moves data from one location to another.

Multiple Output Coil Branches

See **Branch**.

N

Normal Coil

See **Coil, Normal**.

NT

See **Windows NT**.

O

Object

In a GUI environment, this is an element that you can select and manipulate.

Operators

A symbol used in an expression to represent an action. For example, in the expression z + 7 the "+" is an operator representing addition, while the "z" and "7" are operands.

Origin Step

Initial step in a process control Program.

Output

That which comes forth from any computer operation. This may take the form of data, graphics and audio information.

Output Variable

A Variable used for outputting data.

P

Parallel

A method of transmitting several bits of data simultaneously. This can be contrasted with serial which transmits one bit at a time.

Parallel Port

An interface port that supports data transmission of several bits simultaneously. The most common device using this connection is a printer or external removable disc drive. The parallel port is a twenty-five (DB-25) pin connector.

Parameter

A value that, in programming, is passed to a routine. Sometimes known as an argument.

Path

The location of a file in relationship to the various directories and subdirectories that tie it to the root directory.

Platform

The base hardware and software on which a system is built. For instance, an Intel Pentium II Pro processor running Windows NT is a platform, while a Sun Sparc running UNIX is another.

Plug and Play

A PC hardware and operating system specification in which Windows 95/NT automatically detects and addresses PC hardware.

PnP

See **Plug and Play**.

Power Rails

The vertical lines on the left and the right ends of each rung in a Ladder Program. *LogicPro* does not display the right power rails, these are assumed to be there.

Process

This can mean a Program that is running or a task.

Process Flow

The logical "control flow" of a process.

Processor

Microprocessor, the silicon chip that runs the logic for all digital devices and contains the CPU.

Program

Base unit of a Project which defines what the controller does, at what time, and in what sequence.

Program Variables

Variables that are only available at the Program level.

Project

The highest level of organization in *LogicPro.* Projects consist of Resources which, in turn, contain Programs.

Project Variables

Variables that are available to all the Resources and Programs in a Project.

Pull-down menu

A command menu specific to each major *LogicPro* function which drops down from the Menu Bar when selected.

Pulse

A signal characterized by a sharp rise and fall. Either the number of these signals (count) or frequency may be of interest.

R

Radio Button

A type of button used in the graphic user interface that allows you to choose only one option from a group. Selecting an option deselects all of the other options.

Rails

See **Power Rails**.

RAM

An acronym for Random Access Memory, which is used to store variables while a Program is running.

Range

The set of all possible values for a type of variable.

Real

Numerical variables with floating point decimals that can range from -10^{300} to 10^{300} in *LogicPro*.

Rebuild

To recompile a *LogicPro* Program. See **Build**.

Reference Value

A value used for comparison in *LogicPro* Programs.

Resource

Hardware controller which operates the process once the C code generated by the *LogicPro* Program is loaded.

Resource Variables

Variables that are available to all the Programs written for a particular Resource.

Retentive Variable

A variable whose value is saved in the event that power is lost to the controller. Thus, with the restoration of power, this value is still available.

Right Power Rail

This object is not visible in *LogicPro*, it is assumed to be there. See **Power Rail**.

RS-232

Electronics Industry Association (EIA) approved interface for connecting serial devices.

Rubber-Band Line

The graphical indicator of what is being included in a *selection when editing a program*.

Rung

The horizontal line located between the left and right power rails of a Ladder Program where the elements are placed.

S

SCAN

Scan Time Counter Function Block.

Scan Rates

The frequency at which *LogicPro* Programs execute one complete cycle.

Scanning

The time it takes for a *LogicPro* Program to completely execute once.

Select

The act of picking (or highlighting) which object you want to manipulate in a GUI environment.

Selection

A highlighted (or picked) object. See **Select**.

Sequential Function Chart

Graphic way to organize Program using steps to illustrate specific portions of the process, actions to detail what happens inside each step, and transitions to connect the steps into a sequential or parallel flow.

Serial

A method of transmitting data one bit at a time. This can be contrasted with "parallel" which transmits several bits simultaneously.

Serial Port

An interface port that supports transmission of 1 bit at a time. Most PCs are outfitted with serial ports for connecting most peripheral devices (a mouse or a modem). These ports conform to RS-232C or RS-422 standards.

Serial Process

A process in which steps happen in a sequence of one step at a time.

SFC

An acronym for Sequential Function Charts.

Simulation

LogicPro feature which enables the user to verify the operation of a Program before loading and running it on the hardware controller.

Simulation Library

Similar to an Execution Library except that there is no need for an I/O.

Simultaneous Transition

A component of SFC Programs that links and controls execution of steps within the Program.

Standard Toolbar

Screen item below the menu bar displaying buttons which execute commands.

State

A description of the logical time dependent existence of a process.

Status Bar

Item located at the bottom of the screen displaying the current Project and Resource name along with each menu item's information.

Step

An SFC Program unit which follows the instructions and rules defined by its associated actions.

Switch

Electrical element used to open or complete a circuit as desired.

Symbolic Variables

Identifier used to name variables associated with a function block.

System Variable

Variables available at the system level, including statuses of I/O devices, etc.

Symbolic Name

Identifier used to name a variable.

T

Task

Control element which provides periodic and/or triggered execution of a group of associated Program units.

Tag

A variable.

TCP/IP Address

The Transmission Control Protocol/Internet Protocol is a standard for computer network data transmissions developed by the U.S. Department of Defense. The address identifies a unique machine connected to a network.

Text Editor

A Program used to edit text files that does not insert characters other than those defined under ASCII. Although, some older Line editors are still in use today, full screen editors have eclipsed them in popularity. The full screen editor allows you to edit any of the text displayed on the screen. Notepad is an example of a full screen text editor bundled with all Microsoft Windows platforms.

Tile

Arrangement of open Program windows so an equal portion of each window is displayed on the screen at the same time.

TIME

Current Scan Time.

Timer Base

A tool used in simulation to accelerate the testing time.

Tool Bar

LogicPro displays a collection of programming tools along the right side of the screen. The tools available to you depend on the language selected. The toolbar consists of a series of buttons with icons representing different programming functions. See *Chapter 2, The LogicPro Workplace* for a detailed description of all the tools available for each language.

Toolkit

A collection of software "tools" designed to assist in your development and editing tasks.

ToolTips

A user aide that allows you to identify an icon or other desktop element by resting your cursor on it momentarily. The Program responds by opening a small box and displaying the object's name next to the cursor along with a brief explanation in the Status Bar.

Transition

Decision point in an SFC program that determines which step should be active.

Transition Condition

A Boolean expression or variable, the value of which determines whether a transition along a directed link occurs.

Transition Operators

Symbols such as + and – representing mathematical operations used in Transition Conditions.

Trash Can

A folder, or directory, into which deleted items are placed. This allows you to easily recover items that were deleted by mistake. This is a feature available in Windows 95 and NT.

TRUE

In Boolean Logic and Algebra, TRUE and FALSE are the two possible values an expression may have.

Truncate

Omission of trailing digits that do not fit in the available space. This occurs regularly when long floating-point numbers are rounded down, for instance, 3.199999 may be truncated to 3.1.

U

UDFB

See **User Defined Function Block**.

User Defined Function Block

A Program that once defined, compiled and added to the library may be used in Ladder Diagrams and Function Block Diagrams in the same manner that standard and vendor supplied function blocks are used. UDFBs consist of input, output, and internal variables, as well as a set of operations, like a normal function block.

V

Variable

The part of a Program that can be used to store data. Variables can represent the state of physical inputs and outputs, user settings or data, or states internal to the Program.

W

Windows

When spelled with a capital "W" Windows refers to any of the Microsoft Windows programs (Windows 3.1, for Workgroups, Win 95 and NT 4.0). Generically, windows are part of a graphical user interface used to display information and accept input from users.

Windows NT

A 32-bit, preemptive, multitasking Operating System created by Microsoft. Although the NT 4.0 desktop looks very similar to the Windows 95 desktop there are considerable differences between them. Most notable is the higher security and file access available in NT.

Word

A sequence of 16 bits or 2 bytes.